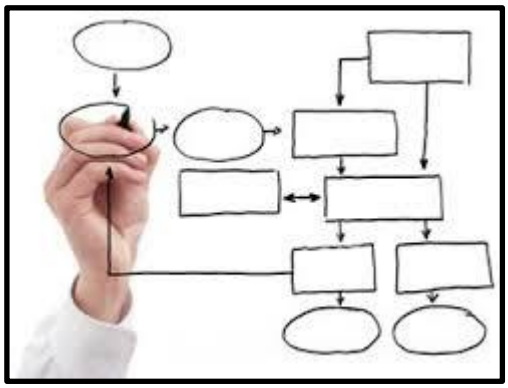


# Unidad 3

## Arquitectura del conjunto de instrucciones.
## Arquitectura de procesadores.
## Interrupción y concurrencia.

# 1- Introducción a los sistemas operativos.

## 1.1 Definición y concepto.

Un Sistema Operativo (SO) es un software que proporciona un acceso sencillo y seguro al soporte físico del ordenador (hardware), ocultando al usuario detalles de la implementación particular y creando la ilusión de existencia de recursos ilimitados (o abundantes). Máquina Virtual.
Otra definición, es el de un programa que actúa como intermediario entre el usuario de la computadora y el hardware de la computadora.El sistema operativo es el programa (o software) más importante de un ordenador. Para que funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras, escáner, etc. En sistemas grandes, el sistema operativo tiene incluso mayor responsabilidad y poder, es como un policía de tráfico, se asegura de que los programas y usuarios que están funcionando al mismo tiempo no interfieran entre ellos. El sistema operativo también es responsable de la seguridad, asegurándose de que los usuarios no autorizados no tengan acceso al sistema.

## 1.2 Funciones y características

Un sistema operativo es un programa que controla la ejecución de los programas de aplicación y que actúa como interfaz entre el usuario de un computador y el hardware de la misma. Puede considerarse que un sistema operativo tiene tres objetivos o lleva a cabo tres funciones:

- Comodidad: Un sistema operativo hace que un computador sea más cómoda de utilizar.
- Eficiencia: Un sistema operativo permite que los recursos de un sistema informático se aprovechen de una manera más eficiente.
- Capacidad de evolución: Un sistema operativo debe construirse de modo que permita el desarrollo efectivo, la verificación y la introducción de nuevas funciones en el sistema y, a la vez, no interferir en los servicios que brinda.

### El Sistema Operativo como Interfaz Usuario/Computadora

El hardware y el software que se utilizan para proveer de aplicaciones a los usuarios pueden contemplarse de forma estratificada o jerárquica. Al usuario de estas aplicaciones se le llama usuario final y, generalmente, no tiene que ocuparse de la arquitectura del computador. Por tanto, el usuario final ve al sistema informático en términos de aplicaciones. Las aplicaciones pueden construirse con un lenguaje de programación y son desarrolladas por programadores de aplicaciones. Si se tuviera que desarrollar un programa de aplicación como un conjunto de instrucciones máquina que sean del todo responsables del control del hardware, se tendría una tarea abrumadora y compleja. Para facilitar esta tarea, se ofrecen una serie de programas de sistemas. Algunos de estos programas se denominan

utilidades e implementan funciones muy utilizadas que ayudan a la creación de los programas, la gestión de los archivos y el control de los dispositivos de E/S. Los programadores hacen uso de estos servicios en el desarrollo de una aplicación y ésta, mientras se está ejecutando, invoca a estas utilidades para llevar a cabo ciertas acciones. El programa de sistemas más importante es el sistema operativo. El sistema operativo oculta al programador los detalles del hardware y le proporciona una interfaz cómoda para utilizar el sistema. Actúa como mediador, facilitándole al programador y a los programas de aplicación el acceso y uso de todas esas características y servicios.
De forma resumida, un sistema operativo ofrece servicios en las áreas siguientes:

- **Creación de programas:** El sistema operativo ofrece una variedad de características y servicios, tales como los editores y los depuradores (debuggers), para ayudar al programador en la creación de programas. Normalmente, estos servicios están en forma de programas de utilidad que no forman realmente parte del sistema operativo, pero que son accesibles a través del mismo.
- **Ejecución de programas:** Para ejecutar un programa se necesita un cierto número de tareas. Las instrucciones y los datos se deben cargar en la memoria principal, los archivos y los dispositivos de E/S se deben inicializar y se deben preparar otros recursos. El sistema operativo administra todas estas tareas para el usuario.


**Figura 1. Niveles y vistas de un sistema informático**

- **Acceso a los dispositivos de E/S:** Cada dispositivo de E/S requiere un conjunto propio y peculiar de instrucciones o de señales de control para su funcionamiento. El sistema operativo tiene en cuenta estos detalles de modo que el programador pueda pensar en forma de lecturas y escrituras simples.
- **Acceso controlado a los archivos:** En el caso de los archivos, el control debe incluir una comprensión, no sólo de la naturaleza del dispositivo de E/S (controlador de disco, controlador de cinta) sino del formato de los archivos y del medio de almacenamiento. Una vez más, es el sistema operativo el que se encarga de los detalles. Es más, en el caso de sistemas con varios usuarios trabajando simultáneamente, es el sistema operativo el que brinda los mecanismos de control para controlar el acceso a los archivos.
- **Acceso al sistema:** En el caso de un sistema compartido o público, el sistema operativo controla el acceso al sistema como un todo y a los recursos específicos del sistema. Las funciones de acceso pueden brindar protección, a los recursos y a los datos, ante usuarios no autorizados y debe resolver los conflictos en la propiedad de los recursos.

- **Detección y respuesta a errores:** Cuando un sistema informático está en funcionamiento pueden producirse varios errores. Entre estos se incluyen los errores internos y externos del hardware, tales como los errores de memoria, fallos o mal funcionamiento de dispositivos y distintos tipos de errores de software, como el desbordamiento aritmético, el intento de acceder a una posición prohibida de memoria y la incapacidad del sistema operativo para satisfacer la solicitud de una aplicación. En cada caso, el sistema operativo debe dar una respuesta que elimine la condición de error con el menor impacto posible sobre las aplicaciones que están en ejecución. La respuesta puede ser desde terminar el programa que produjo el error, hasta reintentar la operación o, simplemente, informar del error a la aplicación.
- **Contabilidad:** Un buen sistema operativo debe recoger estadísticas de utilización de los diversos recursos y supervisar los parámetros de rendimiento tales como el tiempo de respuesta. Para cualquier sistema, esta información es útil para anticiparse a la necesidad de mejoras futuras y para ajustar el sistema y así mejorar su rendimiento. En un sistema multiusuario, la información puede ser utilizada con propósito de cargar en cuenta.

## El sistema operativo como administrador de recursos

Un computador es un conjunto de recursos para el traslado, almacenamiento y proceso de datos y para el control de estas funciones. El sistema operativo es el responsable de la gestión de estos recursos.
¿Se puede afirmar que es el sistema operativo el que controla el traslado, almacenamiento y proceso de los datos? Desde un punto de vista, la respuesta es afirmativa: Administrando los recursos del computador, el sistema operativo tiene el control sobre las funciones básicas de la misma. Pero este control se ejerce de una manera curiosa. Normalmente, se piensa en un mecanismo de control como algo externo a lo controlado o, al menos, como algo distinto y una parte separada de lo controlado. (Por ejemplo, un sistema de calefacción de una estancia es controlado por un termostato, que es algo completamente diferente de los aparatos de generación de calor y de distribución del calor). Este no es el caso de un sistema operativo, que no es habitual como mecanismo de control en dos aspectos:

- El sistema operativo funciona de la misma manera que el software normal de un computador, es decir, es un programa ejecutado por el procesador.
- El sistema operativo abandona con frecuencia el control y debe depender del procesador para recuperarlo.

El sistema operativo es, de hecho, nada más que un programa del computador. Como otros programas de computador, da instrucciones al procesador. La diferencia clave está en el propósito del programa. El sistema operativo dirige al procesador en el empleo de otros recursos del sistema y en el control del tiempo de ejecución de otros programas. Pero para que el procesador pueda hacer estas cosas, debe cesar la ejecución del programa del sistema operativo y ejecutar otros programas. Así pues, el sistema operativo cede el control al procesador para hacer algún trabajo "útil" y luego lo retoma durante el tiempo suficiente para preparar el procesador para llevar a cabo la siguiente parte del trabajo.
La figura 2 propone los recursos principales que son administrados por el sistema operativo. Una parte del sistema operativo está en la memoria principal. En esta parte está el núcleo (kernel), que incluye las funciones utilizadas con más frecuencia en el sistema operativo y, en un momento dado, puede incluir otras partes del sistema operativo que estén en uso. El resto de la memoria principal contiene datos y otros programas de usuario. Como se verá, la asignación de este recurso (la memoria principal) es controlada conjuntamente por el sistema operativo y por el hardware de gestión de memoria en el procesador. El sistema operativo decide cuándo puede utilizarse un dispositivo de E/S por parte de un programa en ejecución y controla el acceso y la utilización de los archivos. El procesador es, en si mismo, un recurso y es el sistema operativo el que debe determinar cuánto tiempo del procesador debe

dedicarse a la ejecución de un programa de usuario en particular. En el caso de sistemas multiprocesador, la decisión debe distribuirse entre todos los procesadores.

## Facilidad de evolución de un sistema operativo

Un sistema operativo importante evolucionará en el tiempo por una serie de razones:

- **Actualizaciones del hardware y nuevos tipos de hardware:** Por ejemplo, las primeras versiones de UNIX y OS/2 no empleaban mecanismos de paginación, porque funcionaban en máquinas sin hardware de paginación. Las versiones más recientes se han modificado para aprovechar las capacidades de paginación. Además, el empleo de terminales gráficos y terminales de pantalla completa, en lugar de los terminales de líneas, pueden influir en el diseño de los sistemas operativos. Por ejemplo, un terminal de éstos puede permitirle al usuario ver diferentes aplicaciones al mismo tiempo, a través de “ventanas” en la pantalla. Esto necesita un soporte más sofisticado en el sistema operativo.
- **Nuevos servicios:** Como respuesta a las demandas del usuario o a las necesidades de los administradores del sistema, el sistema operativo ampliará su oferta de servicios. Por ejemplo, si se determina que es difícil de mantener un buen rendimiento para los usuarios con las herramientas existentes, se deben añadir nuevas medidas y herramientas de control al sistema operativo. Otro ejemplo es el de las nuevas aplicaciones que exigen el uso de ventanas en la pantalla. Esta característica requiere actualizaciones mayores en el sistema operativo.
- **Correcciones:** Desafortunadamente, el sistema operativo tiene fallos que se descubrirán con el curso del tiempo y que es necesario corregir. Por supuesto, estas correcciones pueden introducir nuevos fallos a su vez y así sucesivamente.
- La necesidad de hacer cambios en un sistema operativo de forma regular introduce ciertos requisitos en el diseño. Una afirmación obvia es que el sistema debe tener una construcción modular, con interfaces bien definidas entre los módulos y debe estar bien documentado. Para programas grandes, como normalmente son los sistemas operativos actuales, no es adecuado lo que podría denominarse modularización elemental [DENN80a]. Es decir, debe hacerse mucho más que dividir simplemente un programa en subrutinas. Se volverá a este tema más adelante en el capítulo

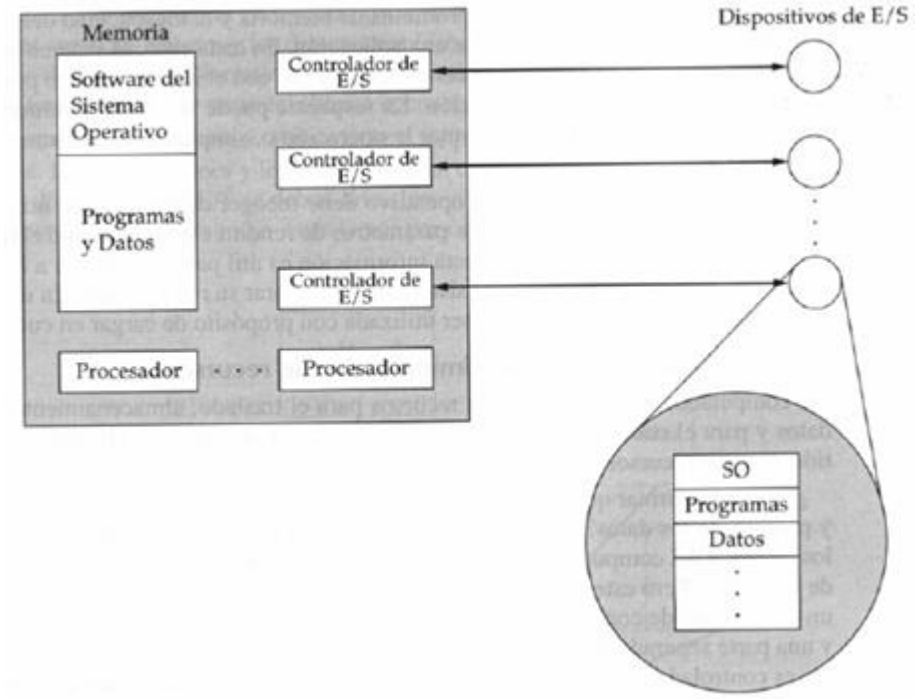


Figura 2 El sistema operativo como administrador de recursos

## 1.3 Evolución histórica

Los sistemas operativos han estado evolucionando durante muchos años. Dado que, históricamente, los sistemas operativos han estado de manera muy estrecha vinculados con la arquitectura de las computadoras en las que se ejecutan, estudiaremos las sucesivas generaciones de computadoras para ver qué clase de sistemas operativos usaban. Esta correspondencia entre las generaciones de sistemas operativos y de computadoras es algo burda, pero establece un poco de estructura que de otra forma sería inexistente.

La primera computadora digital verdadera fue diseñada por el matemático inglés Charles Babbage (1792-1871). Aunque Babbage invirtió la mayor parte de su vida y su fortuna tratando de construir su “máquina analítica”, nunca logró que funcionara correctamente porque era totalmente mecánica, y la tecnología de su época no podía producir las ruedas, engranes y levas con la elevada precisión que él requería. Huelga decir que la máquina analítica no contaba con un sistema operativo.

Como acotación histórica interesante, diremos que Babbage se dio cuenta de que necesitaría software para su máquina analítica, así que contrató a una joven mujer, Ada Lovelace, hija del famoso poeta británico, Lord Byron, como la primera programadora de la historia. El lenguaje de programación Ada recibió su nombre en honor a ella.

**La primera generación (1945-55): Tubos de vacío y tableros de conmutación.**

Después del fracaso de los trabajos de Babbage, fueron pocos los avances que se lograron en la construcción de computadoras digitales hasta la Segunda Guerra Mundial. A mediados de la década de 1940, Howard Aiken en Harvard, John von Neumann en el Institute for Advanced Study en Princeton, J. Presper Eckert y William Mauchley en la University of Pennsylvania y Konrad Zuse en Alemania, entre otros, lograron construir máquinas calculadoras usando tubos de vacío. Estas máquinas eran enormes, y ocupaban cuartos enteros con decenas de miles de tubos de vacío, pero eran mucho más lentas que incluso las computadoras personales más baratas de la actualidad.

En esos primeros días, un solo grupo de personas diseñaba, construía, programaba, operaba y mantenía a cada máquina. Toda la programación se realizaba en lenguaje de máquina absoluto, a menudo alambrando tableros de conmutación para controlar las funciones básicas de la máquina. No existían los lenguajes de programación (ni siquiera los de ensamblador). Nadie había oído hablar de los sistemas operativos. La forma de operación usual consistía en que el programador se anotaba para recibir un bloque de tiempo en la hoja de reservaciones colgada en la pared, luego bajaba al cuarto de la máquina, insertaba su tablero de conmutación en la computadora, y pasaba las siguientes horas con la esperanza de que ninguno de los cerca de 20000 tubos de vacío se quemara durante la sesión. Prácticamente todos los problemas eran cálculos numéricos directos, como la producción de tablas de senos y cosenos.

A principios de la década de 1950, la rutina había mejorado un poco con la introducción de las tarjetas perforadas. Ahora era posible escribir programas en tarjetas e introducirlas para ser leídas, en lugar de usar tableros de conmutación; por lo demás, el procedimiento era el mismo.

**La segunda generación (1955-65): Transistores y sistemas por lote:**

La introducción del transistor a mediados de la década de 1950 alteró el panorama radicalmente. Las computadoras se hicieron lo bastante confiables como para poderse fabricar y vender a clientes comerciales con la expectativa de que seguirían funcionando el tiempo suficiente para realizar algo de trabajo útil. Por primera vez, había una separación clara entre diseñadores, constructores, operadores, programadores y personal de mantenimiento.

Estas máquinas se encerraban en cuartos de computadora con acondicionamiento de aire especial, con equipos de operadores profesionales para operarias. Sólo las grandes empresas, o las principales dependencias del gobierno o universidades, podían solventar el costo de muchos millones de dólares. Para ejecutar un **trabajo** (es decir, un programa o serie de programas), un programador escribía primero

el programa en papel (en FORTRAN o ensamblador) y luego lo perforaba en tarjetas. Después, llevaba el grupo de tarjetas al cuarto de entrada y lo entregaba a uno de los operadores.
Cuando la computadora terminaba el trabajo que estaba ejecutando en ese momento, un operador acudía a la impresora, separaba la salida impresa y la llevaba al cuarto de salida donde el programador podía recogerla después. Luego, el operador tomaba uno de los grupos de tarjetas traídos del cuarto de entrada y lo introducía en el lector. Si se requería el compilador de FORTRAN, el operador tenía que traerlo de un archivero e introducirlo en el lector. Gran parte del tiempo de computadora se desperdiciaba mientras los operadores iban de un lugar a otro, en el cuarto de la máquina.
Dado el alto costo del equipo, no es sorprendente que la gente pronto buscara formas de reducir el desperdicio de tiempo. La solución que se adoptó generalmente fue el **sistema por lotes**. El principio de este modo de operación consistía en juntar una serie de trabajos en el cuarto de entrada, leerlos y grabarlos en una cinta magnética usando una computadora pequeña y (relativamente) económica, como una IBM 1401, que era muy buena para leer tarjetas, copiar cintas e imprimir salidas, pero no para realizar cálculos numéricos. Otras máquinas, mucho más costosas, como la IBM 7094, se usaban para la computación propiamente dicha. Esta situación se muestra en la Fig. 3.

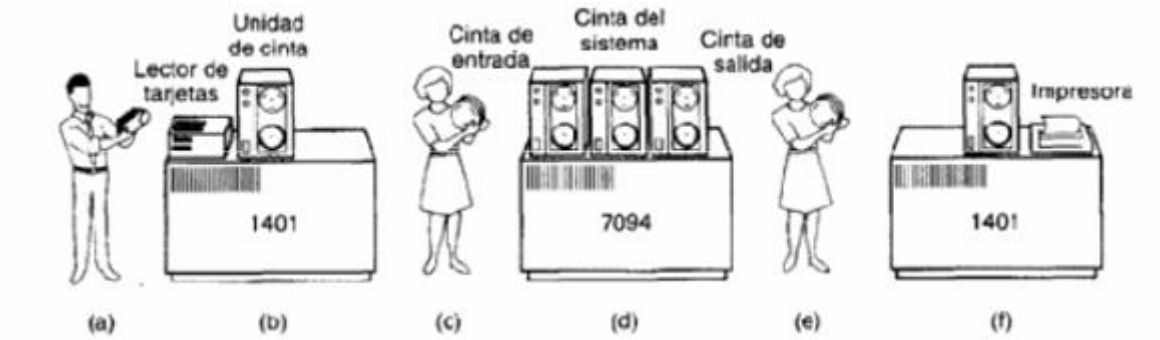



Figura 3 Uno de los primeros sistemas por lotes. (a) Los programadores traen tarjetas a la 1401. (b) La 1401 lee lotes de trabajos y los graba en cinta. (c) El operador lleva la cinta de entrada a la 7094. (d) La 7094 realiza la computación. (e) El operador lleva la cinta salida a la 1401. (f) la 1401 imprime la salida.

Después de cerca de una hora de reunir un lote de trabajos, la cinta se rebobinaba y se llevaba al cuarto de la máquina, donde se montaba en una unidad de cinta. El operador cargaba entonces un programa especial (el antepasado del sistema operativo actual), que leía el primer trabajo de la cinta y lo ejecutaba. La salida se escribía en una segunda cinta, en lugar de imprimirse. Cada vez que terminaba un trabajo, el sistema operativo leía automáticamente el siguiente trabajo de la cinta y comenzaba a ejecutarlo. Una vez que estaba listo todo el lote, el operador desmontaba las cintas de entrada y salida, montaba la cinta de entrada del siguiente lote, y llevaba la cinta de salida a una 1401 para la impresión **fuera de línea** (o sea, no conectada a la computadora principal).
La estructura de un trabajo de entrada típico se muestra en la Fig. 4. El trabajo comenzaba con una tarjeta $JOB, que especificaba el tiempo de ejecución máximo en minutos, el número de cuenta al que se debía cobrar el trabajo, y el nombre del programador. Luego venía una tarjeta $FORTRAN, que ordenaba al sistema operativo leer el compilador de FORTRAN de la cinta de sistema. Esta tarjeta iba seguida del programa por compilar y por una tarjeta $LOAD, que ordenaba al sistema operativo cargar el programa objeto recién compilado. (Los programas compilados a menudo se escribían en cintas temporales y tenían que cargarse explícitamente.) Luego venía la tarjeta $RUN, que ordenaba al sistema operativo ejecutar el programa con los datos que le seguían. Por último, la tarjeta $END marcaba el final del trabajo. Estas tarjetas de control primitivas eran los precursores de los lenguajes de control de trabajos e intérpretes de comandos modernos.

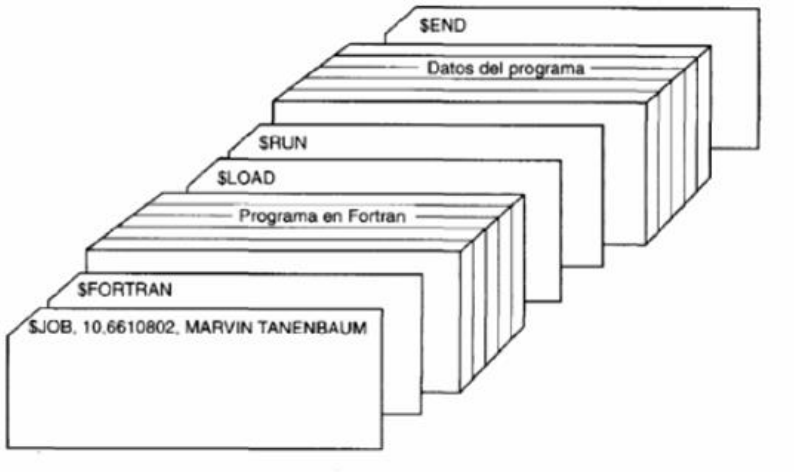



Figura 4. Estructura de un trabajo FMS típico.

Las computadoras grandes de la segunda generación se usaban primordialmente para cálculos científicos y de ingeniería, como la resolución de ecuaciones diferenciales parciales. Estas máquinas generalmente se programaban en FORTRAN y lenguaje ensamblador. Los sistemas operativos típicos eran FMS (el Fortran Monitor System) e IBSYS, el sistema operativo de IBM para la 7094.

## 1.4 Clasificación.

Con el paso del tiempo, los Sistemas Operativos fueron clasificándose de diferentes maneras, dependiendo del uso o de la aplicación que se les daba. A continuación se mostrarán diversos tipos de Sistemas Operativos que existen en la actualidad, con algunas de sus características:

**Sistemas Operativos de multiprogramación (o Sistemas Operativos de multitarea).**

Es el modo de funcionamiento disponible en algunos sistemas operativos, mediante el cual una computadora procesa varias tareas al mismo tiempo. Existen varios tipos de multitareas. La conmutación de contextos (context Switching) es un tipo muy simple de multitarea en el que dos o más aplicaciones se cargan al mismo tiempo, pero en el que solo se está procesando la aplicación que se encuentra en primer plano (la que ve el usuario). Para activar otra tarea que se encuentre en segundo plano, el usuario debe traer al primer plano la ventana o pantalla que contenga esa aplicación. En la multitarea cooperativa, la que se utiliza en el sistema operativo Macintosh, las tareas en segundo plano reciben tiempo de procesado durante los tiempos muertos de la tarea que se encuentra en primer plano (por ejemplo, cuando esta aplicación esta esperando información del usuario), y siempre que esta aplicación lo permita. En los sistemas multitarea de tiempo compartido, como OS/2, cada tarea recibe la atención del microprocesador durante una fracción de segundo. Para mantener el sistema en orden, cada tarea recibe un nivel de prioridad o se procesa en orden secuencial. Dado que el sentido temporal del usuario es mucho más lento que la velocidad de procesamiento del ordenador, las operaciones de multitarea en tiempo compartido parecen ser simultáneas.

Se distinguen por sus habilidades para poder soportar la ejecución de dos o más trabajos activos (que se están ejecutado) al mismo tiempo. Esto trae como resultado que la Unidad Central de Procesamiento (UCP) siempre tenga alguna tarea que ejecutar, aprovechando al máximo su utilización.

Su objetivo es tener a varias tareas en la memoria principal, de manera que cada uno está usando el procesador, o un procesador distinto, es decir, involucra máquinas con más de una UCP.

Sistemas Operativos como UNIX, Windows 95, Windows 98, Windows NT, MACOS, OS/2, soportan la multitarea.
Las características de un Sistema Operativo de multiprogramación o multitarea son las siguientes:

- Mejora productividad del sistema y utilización de recursos.
- Multiplexa recursos entre varios programas.
- Generalmente soportan múltiples usuarios (multiusuarios).
- Proporcionan facilidades para mantener el entorno de usuarios individuales.
- Requieren validación de usuario para seguridad y protección.
- Proporcionan contabilidad del uso de los recursos por parte de los usuarios.
- Multitarea sin soporte multiusuario se encuentra en algunos computadores personales o en sistemas de tiempo real.
- Sistemas multiprocesadores son sistemas multitareas por definición ya que soportan la ejecución simultánea de múltiples tareas sobre diferentes procesadores.
- En general, los sistemas de multiprogramación se caracterizan por tener múltiples programas activos compitiendo por los recursos del sistema: procesador, memoria, dispositivos periféricos.

**Sistema Operativo Monotareas.**

Los sistemas operativos monotareas son más primitivos y es todo lo contrario al visto anteriormente, es decir, solo pueden manejar un proceso en cada momento o que solo puede ejecutar las tareas de una en una. Por ejemplo cuando la computadora está imprimiendo un documento, no puede iniciar otro proceso ni responder a nuevas instrucciones hasta que se termine la impresión.

**Sistema Operativo Monousuario.**

Los sistemas monousuarios son aquellos que nada más puede atender a un solo usuario, gracias a las limitaciones creadas por el hardware, los programas o el tipo de aplicación que se este ejecutando.

Estos tipos de sistemas son muy simples, porque todos los dispositivos de entrada, salida y control dependen de la tarea que se está utilizando, esto quiere decir, que las instrucciones que se dan, son procesadas de inmediato; ya que existe un solo usuario y están orientados principalmente por los microcomputadores.

**Sistema Operativo Multiusuario.**

Es todo lo contrario a monousuario; y en esta categoría se encuentran todos los sistemas que cumplen simultáneamente las necesidades de dos o más usuarios, que comparten mismos recursos. Este tipo de sistemas se emplean especialmente en redes.

En otras palabras consiste en el fraccionamiento del tiempo (timesharing).

**Sistemas Operativos por lotes.**

Los Sistemas Operativos por lotes, procesan una gran cantidad de trabajos con poca o ninguna interacción entre los usuarios y los programas en ejecución. Se reúnen todos los trabajos comunes para realizarlos al mismo tiempo, evitando la espera de dos o más trabajos como sucede en el procesamiento

en serie. Estos sistemas son de los más tradicionales y antiguos, y fueron introducidos alrededor de 1956 para aumentar la capacidad de procesamiento de los programas.

Cuando estos sistemas son bien planeados, pueden tener un tiempo de ejecución muy alto, porque el procesador es mejor utilizado y los Sistemas Operativos pueden ser simples, debido a la secuenciabilidad de la ejecución de los trabajos.

Algunos ejemplos de Sistemas Operativos por lotes exitosos son el SCOPE, del DC6600, el cual está orientado a procesamiento científico pesado, y el EXEC II para el UNIVAC 1107, orientado a procesamiento académico.

Algunas otras características con que cuentan los Sistemas Operativos por lotes son:

- Requiere que el programa, datos y órdenes al sistema sean remitidos todos juntos en forma de lote.
- Permiten poca o ninguna interacción usuario/programa en ejecución.
- Mayor potencial de utilización de recursos que procesamiento serial simple en sistemas multiusuarios.
- No conveniente para desarrollo de programas por bajo tiempo de retorno y depuración fuera de línea.
- Conveniente para programas de largos tiempos de ejecución (ej, análisis estadísticos, nóminas de personal, etc.).
- Se encuentra en muchos computadores personales combinados con procesamiento serial.
- Planificación del procesador sencilla, típicamente procesados en orden de llegada.
- Planificación de memoria sencilla, generalmente se divide en dos: parte residente del S.O. y programas transitorios.
- No requieren gestión crítica de dispositivos en el tiempo.
- Suelen proporcionar gestión sencilla de manejo de archivos: se requiere poca protección y ningún control de concurrencia para el acceso.

**Sistemas Operativos de tiempo real.**

Los Sistemas Operativos de tiempo real son aquellos en los cuales no tiene importancia el usuario, sino los procesos. Por lo general, están subutilizados sus recursos con la finalidad de prestar atención a los procesos en el momento que lo requieran. Se utilizan en entornos donde son procesados un gran número de sucesos o eventos.

Muchos Sistemas Operativos de tiempo real son construidos para aplicaciones muy específicas como control de tráfico aéreo, bolsas de valores, control de refinerías, control de laminadores. También en el ramo automovilístico y de la electrónica de consumo, las aplicaciones de tiempo real están creciendo muy rápidamente. Otros campos de aplicación de los Sistemas Operativos de tiempo real son los siguientes:

- Control de trenes.
- Telecomunicaciones.
- Sistemas de fabricación integrada.
- Producción y distribución de energía eléctrica.
- Control de edificios.
- Sistemas multimedia.

Algunos ejemplos de Sistemas Operativos de tiempo real son: VxWorks, Solaris, Lyns OS y Spectra. Los Sistemas Operativos de tiempo real, cuentan con las siguientes características:

- Se dan en entornos en donde deben ser aceptados y procesados gran cantidad de sucesos, la mayoría externos al sistema computacional, en breve tiempo o dentro de ciertos plazos.
- Se utilizan en control industrial, conmutación telefónica, control de vuelo, simulaciones en tiempo real., aplicaciones militares, etc.
- Objetivo es proporcionar rápidos tiempos de respuesta.
- Procesa ráfagas de miles de interrupciones por segundo sin perder un solo suceso.
- Proceso se activa tras ocurrencia de suceso, mediante interrupción.
- Proceso de mayor prioridad expropia recursos.
- Por tanto generalmente se utiliza planificación expropiativa basada en prioridades.
- Gestión de memoria menos exigente que tiempo compartido, usualmente procesos son residentes permanentes en memoria.
- Población de procesos estática en gran medida.
- Poco movimiento de programas entre almacenamiento secundario y memoria.
- Gestión de archivos se orienta más a velocidad de acceso que a utilización eficiente del recurso.

**Sistemas Operativos de tiempo compartido.**

Permiten la simulación de que el sistema y sus recursos son todos para cada usuario. El usuario hace una petición a la computadora, esta la procesa tan pronto como le es posible, y la respuesta aparecerá en la terminal del usuario.

Los principales recursos del sistema, el procesador, la memoria, dispositivos de E/S, son continuamente utilizados entre los diversos usuarios, dando a cada usuario la ilusión de que tiene el sistema dedicado para sí mismo. Esto trae como consecuencia una gran carga de trabajo al Sistema Operativo, principalmente en la administración de memoria principal y secundaria.

Ejemplos de Sistemas Operativos de tiempo compartido son Multics, OS/360 y DEC-10.

Características de los Sistemas Operativos de tiempo compartido:

- Populares representantes de sistemas multiprogramados multiusuario, ej: sistemas de diseño asistido por computador, procesamiento de texto, etc.
- Dan la ilusión de que cada usuario tiene una máquina para sí.
- Mayoría utilizan algoritmo de reparto circular.
- Programas se ejecutan con prioridad rotatoria que se incrementa con la espera y disminuye después de concedido el servicio.
- Evitan monopolización del sistema asignando tiempos de procesador (time slot).
- Gestión de memoria proporciona protección a programas residentes.
- Gestión de archivo debe proporcionar protección y control de acceso debido a que pueden existir múltiples usuarios accesando un mismo archivos.

**Sistemas Operativos distribuidos.**

Permiten distribuir trabajos, tareas o procesos, entre un conjunto de procesadores. Puede ser que este conjunto de procesadores esté en un equipo o en diferentes, en este caso es trasparente para el usuario.

Existen dos esquemas básicos de éstos. Un sistema fuertemente acoplado es a es aquel que comparte la memoria y un reloj global, cuyos tiempos de acceso son similares para todos los procesadores. En un sistema débilmente acoplado los procesadores no comparten ni memoria ni reloj, ya que cada uno cuenta con su memoria local.

Los sistemas distribuidos deben de ser muy confiables, ya que si un componente del sistema se compone otro componente debe de ser capaz de reemplazarlo.

Entre los diferentes Sistemas Operativos distribuidos que existen tenemos los siguientes: Sprite, Solaris-MC, Mach, Chorus, Spring, Amoeba, Taos, etc.

Características de los Sistemas Operativos distribuidos:

- Colección de sistemas autónomos capaces de comunicación y cooperación mediante interconexiones hardware y software .
- Gobierna operación de un S.C. y proporciona abstracción de máquina virtual a los usuarios.
- Objetivo clave es la transparencia.
- Generalmente proporcionan medios para la compartición global de recursos.
- Servicios añadidos: denominación global, sistemas de archivos distribuidos, facilidades para distribución de cálculos (a través de comunicación de procesos internodos, llamadas a procedimientos remotos, etc.).

**Sistemas Operativos de red.**

Son aquellos sistemas que mantienen a dos o más computadoras unidas através de algún medio de comunicación (físico o no), con el objetivo primordial de poder compartir los diferentes recursos y la información del sistema.

El primer Sistema Operativo de red estaba enfocado a equipos con un procesador Motorola 68000, pasando posteriormente a procesadores Intel como Novell Netware.

Los Sistemas Operativos de red mas ampliamente usados son: Novell Netware, Personal Netware, LAN Manager, Windows NT Server, UNIX, LANtastic.

## Sistemas Operativos paralelos.

En estos tipos de Sistemas Operativos se pretende que cuando existan dos o más procesos que compitan por algún recurso se puedan realizar o ejecutar al mismo tiempo.

En UNIX existe también la posibilidad de ejecutar programas sin tener que atenderlos en forma interactiva, simulando paralelismo (es decir, atender de manera concurrente varios procesos de un mismo usuario). Así, en lugar de esperar a que el proceso termine de ejecutarse (como lo haría normalmente), regresa a atender al usuario inmediatamente después de haber creado el proceso.

Ejemplos de estos tipos de Sistemas Operativos están: Alpha, PVM, la serie AIX, que es utilizado en los sistemas RS/6000 de IBM.

# 1.5 Estructura (niveles o estratos de diseño).

**a) cargador**

Cualquier programa que requiere ser ejecutado en la computadora, deberá ser transferido desde su lugar de residencia a la memoria principal.

**b) cargador para el sistema operativo**

Este programa se encarga de transferir desde algún medio de almacenamiento externo (disco, cinta o tambor) a la memoria principal, los programas del sistema operativo que tienen como finalidad establecer el ambiente de trabajo del equipo de cómputo. Existe un programa especial almacenado en memoria ROM que se encarga de accesar a este programa cargador. Cuando el sistema operativo esta cargado en memoria toma el control absoluto de las operaciones del sistema.

**c) cargador incluido en el sistema operativo**

Su función es cargar a memoria todos los archivos necesarios para la ejecución de un proceso.

**Supervisor (ejecutivo o monitor)**

Es el administrador del sistema que controla todo el proceso de la información por medio de un gran número de rutinas que entran en acción cuando son requeridos. Funge como enlace entre los programas del usuario y todas las rutinas que controlan los recursos requeridos por el programa para posteriormente continuar con su ejecución.

El supervisor también realiza otras funciones como son:

- Administra la memoria.
- Administración de las rutinas que controlan el funcionamiento de los recursos de la computadora.
- Manejo de Archivos
- Administración y control de la ejecución de los programas.

**Lenguaje de comunicación**

Es el medio a través del cual el usuario interactúa directamente con el sistema operativo y esta formado por comandos que son introducidos a través de algún dispositivo. Generalmente un comando consta de dos partes, la primera formada por una palabra que identifica el comando y la acción a realizar y la segunda parte por un conjunto de valores o parámetros que permiten seleccionar diversas operaciones de entre los que dispone el comando.

**Utilería de sistema**

Son programas o rutinas del sistema operativo que realizan diversas funciones de uso común o aplicación frecuente como son: clasificar, copiar e imprimir información.

# 1.6 Núcleo.

El Núcleo (o kernel) es una colección de módulos de software que se ejecutan en forma privilegiada lo que significa que tienen acceso pleno a los recursos del sistema. El núcleo normalmente representa sólo una pequeña parte de lo que por lo general se piensa que es todo el sistema operativo, pero es tal vez el código que más se utiliza. Por esta razón, el núcleo reside por lo regular en la memoria principal, mientras que otras partes del sistema operativo son cargadas en la memoria principal sólo cuando se necesitan.

Los núcleos se diseñan para realizar “el mínimo” posible de procesamiento en cada interrupción y dejar que el resto lo realice el proceso apropiado del sistema, que puede operar mientras el núcleo se habilita para atender otras interrupciones.

El núcleo de un sistema operativo normalmente contiene el código necesario para realizar las siguientes funciones:

- Manejo de interrupciones.
- Creación y destrucción de procesos.
- Cambio de estado de los procesos.
- Despacho.
- Suspensión y reanudación de procesos.
- Sincronización de procesos.
- Comunicación entre procesos.
- Manipulación de los bloques de control de procesos.
- Apoyo para las actividades de entrada/salida.
- Apoyo para asignación y liberación de memoria.
- Apoyo para el sistema de archivos.
- Apoyo para el mecanismo de llamada y retorno de un procedimiento.
- Apoyo para ciertas funciones de contabilidad del sistema.

## 1.6.1 Interrupciones.

Casi todos los computadores tienen un mecanismo mediante el cual otros módulos (E/S, memoria) pueden interrumpir la ejecución normal del procesador. La tabla 5. Enumera las clases más comunes de interrupciones.
Las interrupciones aparecen, principalmente, como una vía para mejorar la eficiencia del procesamiento. Por ejemplo, la mayoría de los dispositivos externos son mucho más lentos

| | |
|---|---|
| De programa | Generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, como el desbordamiento aritmético, la división por cero, el intento de ejecutar una instrucción ilegal de la máquina o una referencia a una zona de memoria fuera del espacio permitido al usuario. |
| De reloj | Generadas por un reloj interno del procesador. Esto permite al sistema operativo llevar a cabo ciertas funciones con determinada regularidad. |
| De E/S | Generadas por un controlador de E/S, para indicar que una operación ha terminado normalmente o para indicar diversas condiciones de error. |
| Por fallo del hardware | Generadas por fallos tales como un corte de energía o un error de paridad de la memoria. |

TABLA 1. Clases de Interrupciones

Con las interrupciones, el procesador se puede dedicar a la ejecución de otras instrucciones mientras una operación de E/S está en proceso. Ejemplo: El programa de usuario alcanza un punto en el que hace una llamada al sistema en forma de una llamada ESCRIBIR. El programa de E/S que se invoca consta solo del código de preparación y de la orden concreta de E/S. Después de que se ejecuten estas pocas instrucciones, se devuelve el control al programa de usuario. Mientras tanto, el dispositivo externo estará ocupado recibiendo datos desde la memoria del computador e imprimiéndolos. Esta operación de E/S se lleva a cabo concurrentemente con la ejecución de las instrucciones del programa de usuario.

Cuando el dispositivo de E/S esté disponible, es decir, cuando esté preparado para aceptar más datos desde el procesador, el módulo de E/S de dicho dispositivo enviará una señal de solicitud de interrupción al procesador. El procesador responde suspendiendo la operación del programa en curso y saltando a un programa que da servicio al dispositivo de E/S en par-ticular, conocido como rutina de tratamiento de la interrupción (interrupt handler), reanu-dando la ejecución original después de haber atendido al dispositivo. el instante en que se produce tal interrupción viene indicado con un asterisco **(*).**

Desde el punto de vista del programa de usuario, una interrupción es solamente eso: una interrupción de la secuencia normal de ejecución. Cuando el tratamiento de la interrupción se termina, la ejecución continúa. Así pues, el programa de usuario no tiene que disponer de ningún código especial para dar cabida a las interrupciones; el procesador y el sistema operativo son los responsables de suspender el programa de usuario y reanudarlo después en el mismo punto.

## 1.6.2 Despachador (Scheduler).

OBJETIVO PRINCIPAL DEL DESPACHADOR

Optimizar la eficiencia del sistema, de acuerdo con criterios considerados importantes para el ambiente del sistema operativo.

FUNCIONES

- El despachador examina la prioridad de los procesos.
- Controla los recursos de una computadora y los asigna entre los usuarios.
- Permite a los usuarios correr sus programas.
- Controla los dispositivos de periféricos conectados a la máquina.
- Cambio de contexto.
- Cambio a modo usuario.

Tipos de despachadores

- ***De largo plazo*.** El despachador de largo plazo, cuando existe, trabaja con la cola de los trabajos en lotes y selecciona el siguiente trabajo de lotes a ejecutarse. Su objetivo principal es proporcionar una mezcla balanceada de trabajos al despachador de corto plazo. Este tipo de despachador es invocado cada vez que un proceso termina y abandona el sistema. Su utilización es relativamente poco frecuente. En términos del diagrama de transición de estados de un proceso, el despachador de largo plazo se encarga de la transición de un proceso del estado de dormido al estado de listo.
- ***De mediano plazo***. Cuando existen procesos que necesitan un uso intensivo de las facilidades de entrada y salida, y que por ello permanezcan suspendidos, puede ser que éstos procesos se quiten temporalmente de memoria principal y se guarden en memoria secundaria, hasta que su condición de espera haya concluido (a esta actividad se le conoce como ]””swapping”), para darle oportunidad a otros procesos que quieran ser admitidos. El despachador de mediano plazo se

encarga del manejo de procesos que temporalmente se han enviado a memoria secundaria. En términos del diagrama de transición de estados, el despachador de mediano plazo se encarga de la transición suspendido a listo.

- **De corto plazo.** El despachador de corto plazo asigna el CPU entre los procesos listos en memoria principal. Su objetivo principal es maximizar la eficiencia del sistema de acuerdo con ciertos criterios. Ya que se encarga de las transiciones de listo a ejecutándose. En la práctica, el despachador de corto plazo se invoca cada vez que ocurre un evento que modifique el estado global del sistema. Algunos eventos que provocan tales cambios son:

    - Pulsos de reloj (interrupciones de tiempo)
    - Interrupciones y terminaciones de E/S
    - La mayoría de los llamados operacionales al sistema operativo
    - Envío y recepción de señales
    - Activación de programas interactivos

### 1.6.3 Primitivas de comunicación (IPC).

Es una función básica de los Sistemas operativos.

Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de IPC.

La IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí. Normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red subyacente.

La comunicación se establece siguiendo una serie de reglas (protocolos de comunicación). Los protocolos desarrollados para internet son los mayormente usados: protocolo de internet (capa de red), protocolo de control de transmisión (capa de transporte) y protocolo de transferencia de archivos, protocolo de transferencia de hipertexto (capa de aplicación).

# 2. Administración de procesos y del procesador.

## 2.1 Concepto de proceso.

El concepto de proceso es fundamental en la estructura de los sistemas operativos. Este término fue acuñado por primera vez por los diseñadores de Multics en los años 60. Es un término algo más general que el de trabajo. Se han dado muchas definiciones para el término proceso, entre las que se incluyen las siguientes:

- Un programa en ejecución
- El "espíritu animado" de un programa
- La entidad que puede ser asignada al procesador y ejecutada por él.

## 2.2 Estados y transiciones de los procesos

El estado de un proceso puede plasmarse como un gráfico el cual asemeja una máquina virtual, así por ejemplo sea la siguiente figura que especifica el estatus de un proceso:

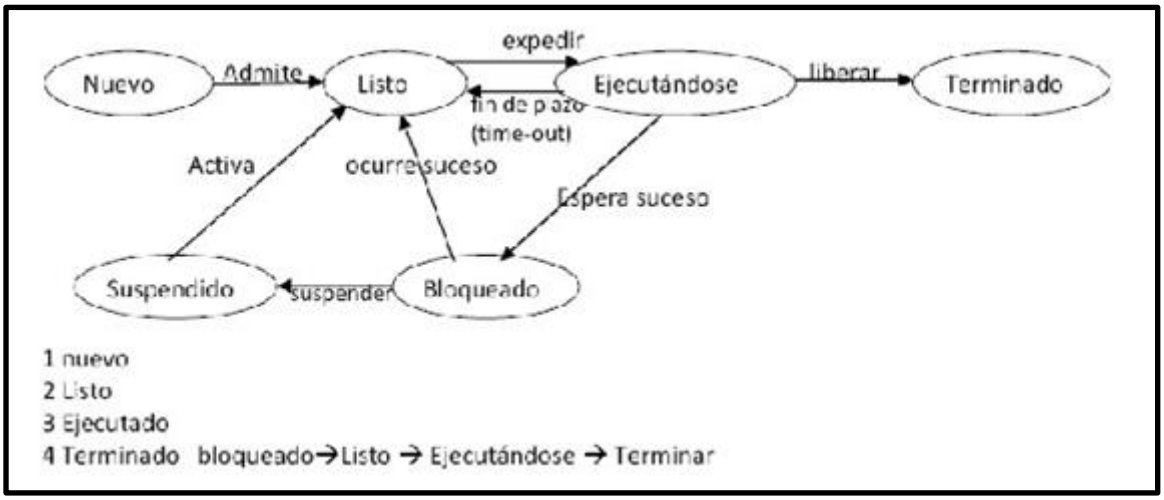



Figura 5. Estados y transiciones de un proceso

Así por ejemplo cuando ninguno de los procesos en memoria principal está en estado Listo el sistema operativo expulsa hacia el disco a alguno de los procesos que este Bloqueado y lo pasa a alguna lista de Suspendidos.

### Transiciones.

**Nuevo–>Listo:** Al crearse un proceso pasa inmediatamente al est

**Listo–>Ejecutando:** En el estado de listo, el proceso solo espera para que se le asigne un procesador para ejecutar (tener en cuenta que puede existir más de un procesador en el sistema).Al liberarse un procesador el planificador ( *scheduler*) selecciona el próximo proceso, según algún criterio definido,a ejecutar.

**Ejecutando–>Listo:** Ante una interrupción que se generé, el proceso puede perder el recurso procesador y pasar al estado de listo. El planificador será el encargado de seleccionar el próximo proceso a ejecutar.

**Ejecutando–>Bloqueado:** A medida que el proceso ejecuta instrucciones realiza pedidos en distintos componentes (ej.:genera un pedido de E/S). Teniendo en cuenta que el pedido puede demorar y, además, si está en un sistema multiprogramado, el proceso es puesto en una cola de espera hasta que se complete su pedido. De esta forma, se logra utilizar en forma más eficiente el procesador.

**Bloqueado–>Listo:** Una vez que ocurre el evento que el proceso estaba esperando en la cola de espera, el proceso es puesto nuevamente en la cola de procesos listos.

**Ejecutando->Terminado:** Cuando el proceso ejecuta su última instrucción pasa al estado terminado. El sistema libera las estructuras que representan al proceso.

# 3.- Conceptos de arquitectura de computación.

A la colección de todas las instrucciones que puede ejecutar un procesador se le denomina conjunto de instrucciones y a la completa descripción de dicho conjunto, arquitectura del conjunto de instrucciones.
La arquitectura del conjunto de instrucciones determina cómo debe ser el hardware del procesador y cómo debe estar organizado. Por ello, los manuales de usuario de los procesadores se enfocan principalmente en la descripción del conjunto de instrucciones que el procesador puede ejecutar, así como de los elementos del procesador que son accesibles al programador.
Las instrucciones se definen y almacenan en la memoria del procesador en lenguaje binario, lo cual constituye el llamado código máquina. El lenguaje que sustituye los códigos de operación binarios y las direcciones por nombres simbólicos se denomina lenguaje ensamblador.
El formato de las instrucciones se representa mediante una caja rectangular simbolizando los bits de la instrucción en binario. Estos bits se dividen en grupos llamados campos:

- Campo de código de operación (opcode), que especifica la operación a realizar.
- Campo de dirección(es), que proporciona direcciones de memoria o de registros.
- Campo de modo, que especifica la forma en que se interpreta el campo de direcciones.

Estudiaremos las instrucciones típicas que se pueden encontrar en procesadores comerciales, así como los diferentes formatos en que pueden presentarse, haciendo especial énfasis en el direccionamiento de los operandos.

## CICLO DE OPERACIÓN BÁSICO DE UN PROCESADOR

La unidad de control de un procesador se diseña para ejecutar cada una de las instrucciones de un programa efectuando la siguiente secuencia de pasos:

1.- Obtener la instrucción de memoria. Almacenarla en un registro de control.
2.- Decodificar la instrucción.
3.- Localizar los operandos empleados en la instrucción.
4.- Obtener de la memoria los operandos (si fuese necesario)
5.- Ejecutar la operación en la ruta de datos.
6.- Almacenar el resultado en un lugar adecuado.
7.- Volver al paso 1 y para procesar la siguiente instrucción.

La unidad de control del procesador consta de un registro especial, el contador de programa PC (Program Counter). Su contenido apunta a la posición de memoria de la instrucción que se va a ejecutar a continuación, y se incrementa cada vez que se lee una instrucción del programa almacenado en la memoria.
La decodificación del paso 2 determina la operación a ejecutar y el modo de direccionamiento de la instrucción.
En el paso 3, los operandos se localizan según el modo de direccionamiento y el campo de direcciones de la instrucción.
El procesador ejecuta la instrucción sobre los operandos, almacena el resultado y regresa al paso 1 para obtener la siguiente instrucción del programa.

## CONJUNTO DE REGISTROS

El conjunto de registros está formado por todos los registros del procesador accesibles al programador:

- El contador de programa PC.
- Banco de registros de la ruta de datos.

- El Puntero de pila SP (Stack Pointer). Permite definir estructuras LIFO en la memoria.
- El registro de estado del procesador PSR (Processor Status Register). Permite almacenar los bits de estado C, N, V y Z de la ALU. Estos bits de estado, normalmente denominados banderines (flags), pueden utilizarse para tomar decisiones que determinen el flujo de ejecución del programa.

Los procesadores constan además de una serie de registros que normalmente no son accesibles al programador:

- El registro de instrucción IR.
- El registro CAR
- Registros ocultos del banco de registros de la ruta de datos. Sólo son accesibles a los microprogramas para, por ejemplo, almacenar resultados parciales durante la ejecución de una instrucción.
- Los registros de pipeline en procesadores segmentados.

# 4. Direccionamiento de operandos.

En una instrucción de manipulación de datos, como por ejemplo la instrucción ADD que vimos en el tema anterior, se necesita especificar tanto los datos que se van a sumar, como el destino del resultado. Esto se conoce como direccionamiento de operandos. Dichos operandos pueden encontrarse tanto en el banco de registros como en la memoria.

Si la dirección del operando forma parte de la instrucción, se tiene un direccionamiento explícito. En caso contrario, se tiene un direccionamiento implícito.

El número de operandos que se pueden direccionar explícitamente en una instrucción es un factor importante a tener en cuenta a la hora de definir la arquitectura del conjunto de instrucciones de un procesador.

Como ejemplo, supongamos que se quiere evaluar la siguiente expresión aritmética:

**X = (A+B)(C+D)**

con instrucciones de tres, dos, una y cero direcciones. Supondremos además que:

- Los operandos están en direcciones de memoria representadas por las letras A, B, C y D. La operación no debe alterar sus contenidos.
- El resultado se almacena en la memoria, en una dirección representada por X.
- Las operaciones aritméticas que se pueden utilizar son la suma ADD y la multiplicación MUL sobre dos operandos.
- Las operaciones de transferencia de datos que se pueden utilizar son LD (de memoria a registro), ST (de registro a memoria) y MOVE (entre registros o posiciones de memoria).

## INSTRUCCIONES DE TRES DIRECCIONES

Programa para el cálculo de X = (A+B)(C+D) utilizando instrucciones de tres direcciones:

| | |
|---|---|
| ADD T1, A, B | M[T1] ← M[A] + M[B] |
| ADD T2, C, D | M[T2] ← M[C] + M[D] |
| MUL X, T1, T2 | M[X] ← M[T1] × M[T2] |

T1 y T2 son posiciones de memoria para almacenar los resultados parciales del cálculo.

El mismo programa puede usar registros como posiciones de almacenamiento temporal:

| | |
|---|---|
| ADD R1, A, B | R1 ← M[A] + M[B] |
| ADD R2, C, D | R2 ← M[C] + M[D] |
| MUL X, R1, R2 | M[X] ← R1 × R2 |

El uso de registros puede reducir el tiempo de ejecución del programa entre 5 y 9 veces.
Ventaja: Se reducen las longitudes de los programas para la evaluación de expresiones.
Inconveniente: El código binario de la instrucción necesita más bits para poder especificar las tres direcciones (sobre todo si son posiciones de memoria).

## INSTRUCCIONES DE DOS DIRECCIONES

En la instrucción sólo hay dos direcciones explícitas (registros o posiciones de memoria). Se supone implícitamente que el resultado de la operación indicada en la instrucción se almacena en la dirección del primer operando indicado explícitamente.
Programa para el cálculo de X = (A+B)(C+D) utilizando instrucciones de dos direcciones:

| | |
|---|---|
| MOVE T1, A | M[T1] ← M[A] |
| ADD T1, B | M[T1] ← M[T1] + M[B] |
| MOVE X, C | M[X] ← M[C] |
| ADD X, D | M[X] ← M[X] + M[D] |
| MUL X, T1 | M[X] ← M[X] × M[T1] |

Al igual que en el caso de las tres direcciones, se pueden usar registros en lugar de las posiciones de memoria para almacenar los cálculos parciales.
Para poder realizar el cálculo deseado ha habido que:

- Introducir operaciones de transferencia de datos MOVE.
- Aumentar la longitud del programa.

Como contrapartida, la longitud de la instrucción es menor.

## INSTRUCCIONES DE UNA DIRECCIÓN

En la instrucción sólo hay una dirección explícita. El procesador utiliza implícitamente un registro acumulador ACC para poder realizar tanto las operaciones aritméticas, siendo la fuente de uno de los operandos y el destino del resultado, como las de transferencia de datos.
Programa para el cálculo de X = (A+B)(C+D) utilizando instrucciones de una dirección:

| | | |
|---|---|---|
| LD | A | ACC ← M[A] |
| ADD | B | ACC ← ACC + M[B] |
| ST | X | M[X] ← ACC |
| LD | C | ACC ← M[C] |
| ADD | D | ACC ← ACC + M[D] |
| MUL | X | ACC ← ACC × M[X] |
| ST | X | M[X] ← ACC |

Ha aumentado el número de líneas de programa y el número de accesos a memoria.

## INSTRUCCIONES DE CERO DIRECCIONES

Para llevar a cabo operaciones aritméticas con cero direcciones, todas ellas deben ser implícitas. El procesador utiliza una estructura LIFO llamada pila (stack), de la que las instrucciones toman los datos y en donde almacenan los resultados:

- TOS (top of stack) es la palabra más arriba en la pila.
- $TOS_{-1}$ es la palabra inmediatamente debajo, $TOS_{-2}$ la siguiente, etc.
- Cuando se usan uno o varios operandos en una operación, se eliminan de la pila. La palabra por debajo pasa a ser la nueva TOS.
- Cuando se genera un resultado, se coloca en la pila pasando a ser el nuevo TOS.

De este modo, TOS y algunas palabras ubicadas por debajo son las direcciones implícitas de los operandos, y TOS es la dirección implícita del resultado. Por ejemplo, la instrucción de suma sería simplemente “ADD”, y la operación que se llevaría a cabo sería:

$$TOS \leftarrow TOS + TOS_{-1}.$$

Para meter y sacar datos de la pila se usan las instrucciones:

- PUSH (empujar). PUSH X transfiere la palabra en la dirección X de la memoria al TOS.
- POP (sacar). POP X transfiere el TOS a la dirección X de la memoria.

Volviendo a nuestro cálculo de ejemplo, el programa con instrucciones de cero direcciones será el siguiente:

| | | Pila | Notación posfijo |
|---|---|---|---|
| PUSH A | TOS ← M[A] | A | A |
| PUSH B | TOS ← M[B] | B<br>A | B |
| ADD | TOS ← TOS + $TOS_{-1}$ | A+B | + |
| PUSH C | TOS ← M[C] | C<br>A+B | C |
| PUSH D | TOS ← M[D] | D<br>C<br>A+B | D |
| ADD | TOS ← TOS + $TOS_{-1}$ | C+D<br>A+B | + |
| MUL | TOS ← TOS × $TOS_{-1}$ | (A+B)(C+D) | × |
| POP X | M[X] ← TOS | | |

Se necesitan en total ocho instrucciones, una más que en el caso de instrucciones de una dirección. Como contrapartida, sólo las instrucciones PUSH y POP utilizan direccionamiento explícito.

# ARQUITECTURAS DE DIRECCIONAMIENTO

Arquitectura memoria a memoria:

- Los operandos se toman directamente de la memoria, y el resultado se envía directamente a la memoria.
- Las instrucciones de manipulación y transferencia de datos contienen entre uno y tres campos de direcciones.
- El procesador sólo tiene registros de control.
- Es el caso del ejemplo del cálculo X=(A+B)(C+D) con instrucciones de tres direcciones:

```
ADD T1, A, B      M[T1] ← M[A] + M[B]
ADD T2, C, D      M[T2] ← M[C] + M[D]
MUL X, T1, T2     M[X] ← M[T1] × M[T2]
```

En la práctica este tipo de arquitecturas no se usa debido a que son lentas:

- Se necesita un elevado número de accesos a memoria, tanto en la fase de búsqueda de las instrucciones del programa como en la de su ejecución.
- Dar la posibilidad de que todas las instrucciones puedan acceder directamente a memoria incrementa la complejidad de las estructuras de control, dando como resultado que el período de la señal de reloj tenga que ser más largo.

Arquitectura registro a registro y carga/almacenamiento:

- Permiten una sola dirección de memoria, y restringen su uso a las instrucciones de carga y almacenamiento.
- Las instrucciones de manipulación de datos acceden solamente a registros. - El procesador necesita un banco de registros con un tamaño adecuado.

En esta arquitectura, el programa ejemplo para el cálculo de X=(A+B)(C+D) será:

| | | | |
|---|---|---|---|
| LD | R1, A | R1 ← M[A] | PUSH A |
| LD | R2, B | R2 ← M[B] | PUSH B |
| ADD | R3, R1, R2 | R3 ← R1 + R2 | ADD |
| LD | R1, C | R1 ← M[C] | PUSH C |
| LD | R2, D | R2 ← M[D] | PUSH D |
| ADD | R1, R1, R2 | R1 ← R1 + R2 | ADD |
| MUL | R1, R1, R3 | R1 ← R1 × R3 | MUL |
| ST | X, R1 | M[X] ← R1 | POP X |

Este tipo de arquitecturas se usa más en la práctica porque, aunque se necesiten más instrucciones en el programa, el número total de accesos a memoria se reduce y, por tanto, son más rápidas.

Arquitectura de memoria-registro:

- Incluyen instrucciones de dos o tres direcciones con una o dos direcciones de memoria.
- La longitud del programa y el número de accesos a memoria tienden a estar en un punto intermedio de las dos arquitecturas anteriores.

Ejemplo de instrucción:

| | | |
|---|---|---|
| ADD | R1, A | R1 ← R1 + M[A] |

Este tipo de arquitectura se usa cuando se quiere dar compatibilidad con programas antiguos.

Arquitectura de un solo acumulador:

- Permiten instrucciones con una sola dirección para acceder a la memoria. - El procesador sólo dispone del acumulador para hacer operaciones.

Es el caso del ejemplo del cálculo X=(A+B)(C+D) con instrucciones de una sola dirección. Esta arquitectura es ineficiente debido a que requiere un gran número de accesos a memoria. Se usa en el caso de aplicaciones sencillas y de bajo coste, que no precisen de un alto rendimiento.

Arquitectura de pila:

- Las instrucciones de manipulación de datos usan una pila de la que obtener los datos y en la que almacenar el resultado.
- Para la transferencia de datos se usan instrucciones de una sola dirección.

Es el caso del ejemplo del cálculo X=(A+B)(C+D) con instrucciones de cero direcciones.

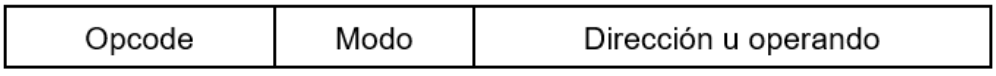
Esta arquitectura resulta especialmente útil para la rápida interpretación de lenguajes de alto nivel, en los que la representación intermedia del código utiliza operaciones con la pila. Para que sea eficiente, el procesador almacena directamente gran parte de las posiciones de la pila.

# 5.- Modos de direccionamiento.

## DEFINICIÓN

El modo de direccionamiento especifica una regla para interpretar o modificar el campo de direcciones de la instrucción antes de que se haga realmente referencia al operando. A la dirección del operando generada mediante la aplicación de esa regla se la denomina dirección efectiva.

Formato de la instrucción:



Los procesadores utilizan técnicas de modo de direccionamiento para ajustarse a las siguientes características:

- Proporcionar flexibilidad al usuario en la programación mediante punteros a memoria, contadores para el control de bucles, indexar datos y reubicar programas.
- Reducir el número de bits de los campos de direcciones de la instrucción.

Disponer de varios modos de direccionamiento proporciona al programador experimentado la posibilidad de escribir programas que requieran pocas instrucciones. Sin embargo, el uso de modos de direccionamiento complejo puede implicar un mayor tiempo de ejecución.

**MODO IMPLÍCITO**

El operando se especifica implícitamente en el código de operación, por lo que la instrucción no necesita de un campo de direcciones.

Ejemplos:

- Operaciones sobre el contenido del acumulador (complementar acumulador).
- Operaciones con la pila (ADD).

**MODO INMEDIATO**

El operando se especifica directamente en la instrucción. En vez de tener un campo de direcciones, la instrucción posee un campo de operando.

Ejemplo: inicialización de un registro con un valor constante.

**MODO REGISTRO**

Se tiene cuando el campo de direcciones de la instrucción especifica un registro del procesador en lugar de una posición de memoria. El operando está contenido en el registro direccionado en la instrucción.

**MODO REGISTRO INDIRECTO**

La instrucción direcciona un registro del procesador en el que está la dirección de memoria donde se encuentra el operando.

Para acceder al operando correctamente, el programador debe asegurarse de que la dirección efectiva esté ya disponible en el registro.

La ventaja de este modo de direccionamiento es que el campo de direcciones de la instrucción necesita menos bits que los que serían necesarios para especificar una dirección de memoria directamente.

**MODO AUTOINCREMENTO (AUTODECREMENTO)**

Este modo es igual al anterior, incluyendo una operación de incremento (decremento) del registro que contiene la dirección de memoria del operando.

Ejemplo:

ADD (R1)+, 3 M[R1] ← M[R1]+3, R1 ← R1+1

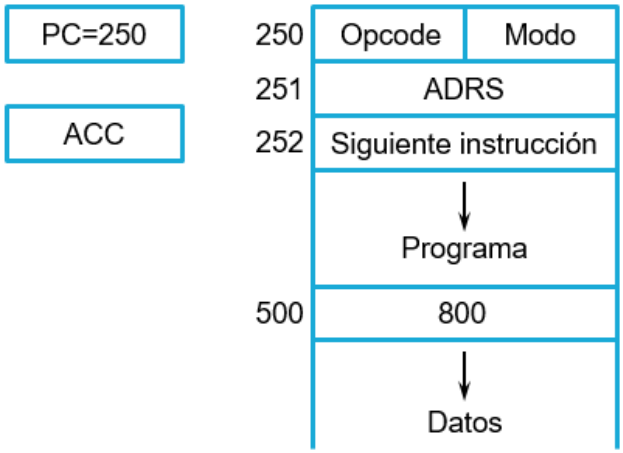
Se utiliza cuando se requiere hacer una misma operación sobre un conjunto de datos (array) almacenado en posiciones consecutivas de la memoria. En el ejemplo anterior, R1 contendrá la dirección del primer dato en el array. Cada vez que se ejecuta la instrucción, se hace la operación sobre el dato y el registro se incrementa para apuntar al siguiente dato del array.

**MODO DE DIRECCIONAMIENTO DIRECTO**

El campo de direcciones de la instrucción proporciona directamente la dirección efectiva.

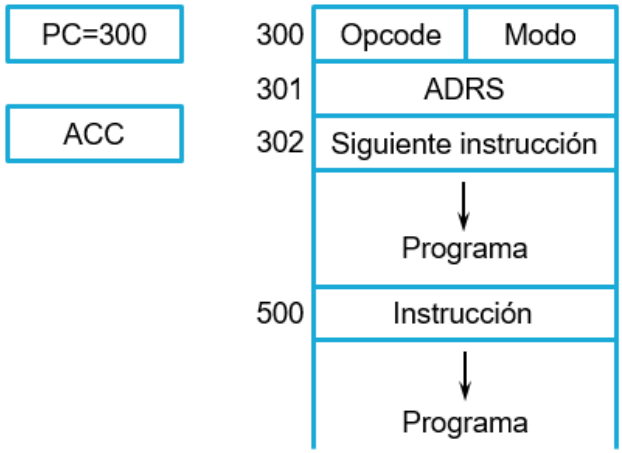
Ejemplos:

(a) Transferencia de datos




Opcode: Carga ACC
Modo: Dirección directa
ADRS: 500
Operación: ACC ← 800
PC ← 252

(b) Salto condicional

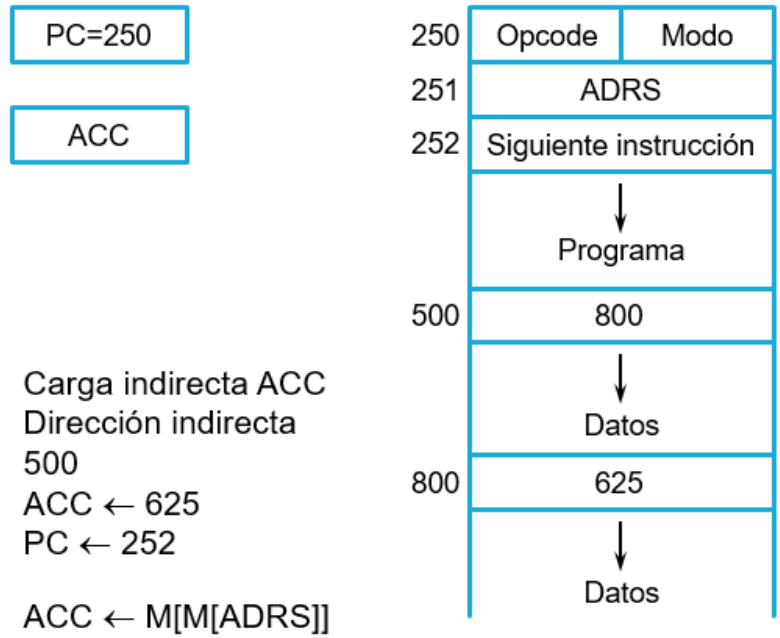



Opcode: Bifurcación si ACC=0
Modo: Dirección directa
ADRS: 500
Operación: PC ← 500 si ACC=0
PC ← 302 si ACC≠0

**MODO DE DIRECCIONAMIENTO INDIRECTO**

El campo de direcciones de la instrucción proporciona la dirección en que se encuentra la dirección efectiva.

Ejemplo: Transferencia de datos



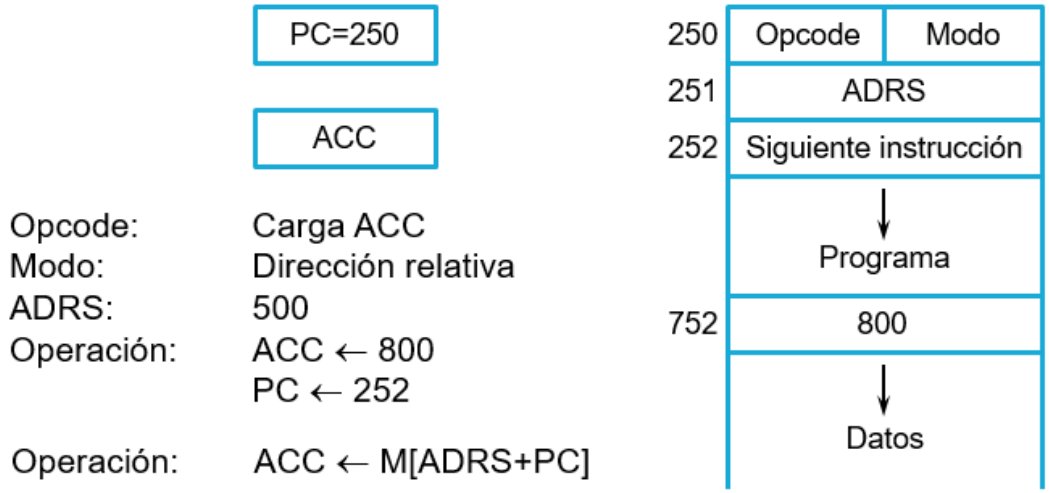

Opcode: Carga indirecta ACC
Modo: Dirección indirecta
ADRS: 500
Operación: ACC ← 625
PC ← 252

Operación: ACC ← M[M[ADRS]]

**MODO DE DIRECCIONAMIENTO RELATIVO**

La dirección efectiva se obtiene como suma de la parte de dirección de la instrucción y el contenido del PC. La parte de dirección de la instrucción es un número con signo.

Ejemplo: Transferencia de datos




Este tipo de direccionamiento se usa frecuentemente en instrucciones de salto condicional cuando la dirección de bifurcación está cercana a la instrucción de salto. Este direccionamiento genera instrucciones más compactas, ya que la dirección relativa necesita de menos bits que una dirección de memoria.

**MODO DE DIRECCIONAMIENTO INDEXADO**

La dirección efectiva se obtiene como suma de la parte de dirección de la instrucción y el contenido de un registro del procesador, llamado registro índice. Este puede ser un registro especial del procesador o simplemente cualquiera de los registros del banco de registros.
Se usa cuando se desea acceder o manipular un array de datos almacenado en la memoria:

- El campo de direcciones de la instrucción marca el comienzo del array.
- Cada operando del array se almacena en memoria en una dirección relativa a su comienzo.
- La distancia entre la dirección de comienzo y la dirección del operando es el valor del índice almacenado en el registro.

Así, con la misma instrucción se puede acceder a cualquier operando del array mediante el uso del registro índice. Si los operandos están consecutivos en la memoria, se accede a cada uno de ellos incrementando el registro índice.

**MODO DE REGISTRO BASE**

La dirección efectiva se obtiene como suma de la parte de dirección de la instrucción y el contenido de un registro del procesador, llamado registro base.
Mientras que en el modo indexado el registro índice contiene un número que es relativo a la parte de dirección de la instrucción, en este modo el registro base es el que contiene la dirección base y el campo de direcciones de la instrucción proporciona el desplazamiento relativo.

## RESUMEN

Para comparar los diferentes modos de direccionamiento, consideraremos como ejemplo una operación de carga del acumulador:

Opcode: Carga ACC
ADRS o NBR: 500

PC=250

R1=400

ACC

| Modo | Nemónico | Transferencias | Dir. ef. | ACC |
|---|---|---|---|---|
| Directo | LDA ADRS | ACC ← M[ADRS] | 500 | 800 |
| Inmediato | LDA #NBR | ACC ← NBR | 251 | 500 |
| Indirecto | LDA [ADRS]<br>LDA @ADRS | ACC ← M[M[ADRS]] | 800 | 300 |
| Relativo | LDA $ADRS | ACC ← M[ADRS+PC] | 752 | 600 |
| Indexado | LDA ADRS (R1) | ACC ← M[ADRS+R1] | 900 | 200 |
| Registro | LDA R1 | ACC ← R1 | — | 400 |
| Registro indirecto | LDA (R1) | ACC ← M[R1] | 400 | 700 |

| | | |
|---|---|---|
| 250 | Opcode | Modo |
| 251 | ADRS o NBR | |
| 252 | Siguiente instrucción | |
| | | |
| 400 | 700 | |
| | | |
| 500 | 800 | |
| | | |
| 752 | 600 | |
| | | |
| 800 | 300 | |
| | | |
| 900 | 200 | |

# 6.- Arquitecturas de conjunto de instrucciones.

## 6.1 ARQUITECTURA RISC

Los procesadores de conjunto reducido de instrucciones RISC (Reduced Instruction Set Computers) tienen las siguientes propiedades:

1. Las instrucciones sólo realizan operaciones elementales.
2. Los accesos a memoria se restringen a las instrucciones de carga y almacenamiento. Las instrucciones de manipulación de datos son de registro a registro.
3. Número limitado de modos de direccionamiento.
4. Los formatos de todas las instrucciones tienen la misma longitud.

Características de la arquitectura RISC:

1. Su objetivo es conseguir un alto rendimiento (alta velocidad de ejecución). Por ello los accesos a memoria se limitan a las instrucciones de carga y almacenamiento.
2. Necesita un banco de registros relativamente grande.
3. Al tener las instrucciones una longitud fija, modos de direccionamiento limitados, y realizar sólo operaciones básicas, la unidad de control es relativamente simple, y típicamente está cableada.
4. La organización interna del procesador es en pipeline.

## 6.2 ARQUITECTURA CISC

Los procesadores de conjunto de instrucciones complejo CISC (Complex Instruction Set Computers) tienen las siguientes propiedades:

1. Las instrucciones realizan tanto operaciones elementales como complejas.
2. Los accesos a memoria están disponibles en prácticamente todos los tipos de instrucciones.
3. Gran número de modos de direccionamiento.
4. Los formatos de las instrucciones son de diferente longitud.

Características de la arquitectura CISC:

1. Su objetivo es proporcionar instrucciones que faciliten realizar programas compactos y así ahorrar memoria.
2. Debido a la alta accesibilidad de la memoria, el banco de registros es reducido.
3. Debido a la complejidad de las instrucciones y la diversidad de sus formatos, la unidad de control es compleja y suele utilizar microprogramación.
4. Internamente, las instrucciones CISC se convierten en una secuencia de instrucciones RISC, siendo procesadas mediante un pipeline tipo RISC.

# 7.- Instrucciones de transferencia de datos.

Las instrucciones de transferencia de datos mueven un dato de un lugar del procesador a otro, sin modificarlo.

Las típicas transferencias son entre memoria y los registros del procesador, entre estos y los registros de entrada/salida (E/S), y entre los propios registros de procesador:

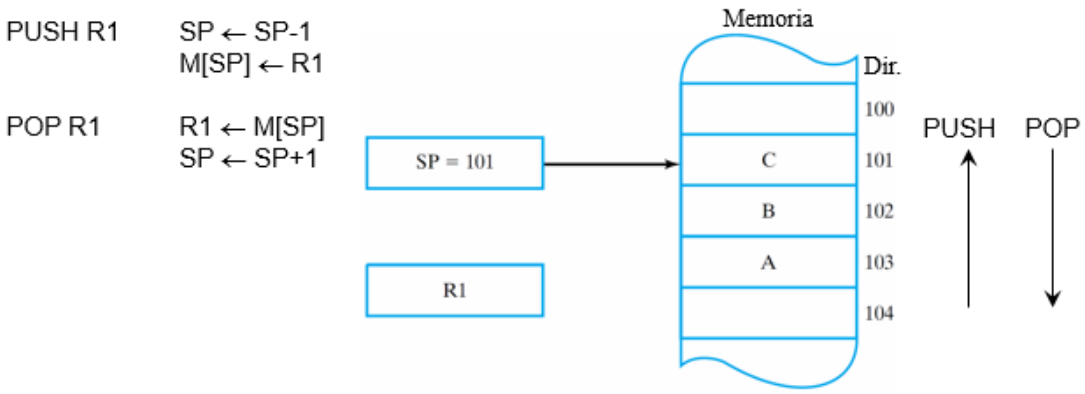
| Nombre | Nemónico | Descripción |
|---|---|---|
| Load | LD | Desde una posición de memoria a un registro del procesador. |
| Store | ST | Desde un registro del procesador a una posición de memoria. |
| Move | MOVE | Desde un registro del procesador a otro. También se utiliza para transferir datos entre dos posiciones de memoria. |
| Exchange | XCH | Intercambia datos entre dos registros del procesador o entre dos posiciones de memoria. |
| Push | PUSH | Transferencia de datos con la pila |
| Pop | POP | |
| Input | IN | Transferencia de datos con registros E/S |
| Output | OUT | |

**TRANSFERENCIA DE DATOS CON LA PILA**

La arquitectura basada en pila posee características que facilitan ciertos tipos de procesamiento de datos y de control de tareas. Así, en algunas calculadoras y procesadores se usan pilas para la evaluación de expresiones aritméticas.

Las pilas en memoria se organizan mediante un registro especial, llamado puntero de pila SP (Stack Pointer). El registro SP contiene la dirección del TOS.

PUSH es la instrucción que almacena datos en la pila, y POP la que los extrae de ella:




SP se inicializa con la dirección de memoria donde quiera iniciarse la pila (104 en el ejemplo).

**TRANSFERENCIA DE DATOS CON REGISTROS E/S**

Los registros E/S, llamados comúnmente puertos, son los que utiliza un procesador para intercambiar datos con dispositivos externos (periféricos). Normalmente cada puerto disponible tendrá asociado un código de dirección.

Las direcciones de los puertos pueden asignarse de dos formas:

a) Sistemas con E/S independiente o separada.

- El rango de direcciones asignadas a la memoria y a los puertos de E/S son independientes entre sí.
- El procesador tiene instrucciones de transferencia de datos diferentes, según se trate de transferencias con la memoria o con los puertos E/S: LD y ST para la memoria, e IN y OUT para los puertos.

b) Sistemas con E/S mapeada o ubicada en memoria.

- Se asigna un subrango de direcciones de memoria para direccionar los puertos E/S: cada puerto E/S se trata como una posición de memoria más.
- El procesador no tiene instrucciones de transferencia de datos específicas para E/S, se usan las mismas que las de la memoria, LD y ST.
- Debido a la complejidad extra que conlleva la E/S independiente, el procesador con E/S mapeada requiere menos lógica interna, es más barato y rápido (tipo RISC).

# 8.- Instrucciones de manipulación de datos.

Las instrucciones de manipulación de datos realizan operaciones sobre los datos, proporcionando los recursos de cálculo del procesador. Son de tres tipos:

1. Instrucciones aritméticas.
2. Instrucciones lógicas y de manipulación de bits.
3. Instrucciones de desplazamiento.

## INSTRUCCIONES ARITMÉTICAS

| Nombre | Nemónico | Descripción |
|---|---|---|
| Incremento | INC | Suma uno. Si todos los bits son 1, genera una palabra con todos los bits a 0. |
| Decremento | DEC | Restar uno. Si todos los bits son 0, genera una palabra con todos los bits a 1. |
| Suma<br>Resta<br>Multiplicación<br>División | ADD<br>SUB<br>MUL<br>DIV | En procesadores sencillos sólo están disponibles las operaciones de suma y de resta. En esos casos, la multiplicación y la división se tienen que hacer mediante programas. Normalmente estas operaciones están disponibles para diferentes tipos de datos. |
| Suma con acarreo | ADDC | Suma dos operandos teniendo en cuenta el acarreo del cálculo anterior. Permite cálculos de doble precisión. |
| Resta con acarreo | SUBB | Resta dos operandos teniendo en cuenta el acarreo (borrow) del cálculo anterior. Permite cálculos de doble precisión. |
| Resta inversa | SUBR | Invierte el orden de los operandos, haciendo la resta B-A en lugar de A-B. |
| Negativo | NEG | Halla el complemento a dos del operando. |

## INSTRUCCIONES LÓGICAS Y DE MANIPULACIÓN DE BITS

Mediante la aplicación de operaciones lógicas, estas instrucciones permiten la manipulación de bits o de grupos de bits en palabras almacenadas en registros o en memoria.
Normalmente, estas instrucciones tratan cada bit del operando individualmente.

| Nombre | Nemónico | Descripción |
|---|---|---|
| Poner a 0 (Clear) | CLR | Pone todos los bits del operando a 0. |
| Poner a 1 (Set) | SET | Pone todos los bits del operando a 1. |
| Complementar | NOT | Complementa los bits del operando. |
| Producto lógico<br>Suma lógica<br>OR exclusiva | AND<br>OR<br>XOR | Realizan la función lógica correspondiente entre dos operandos bit a bit. AND se usa con máscaras para poner selectivamente a 0 algunos bits de los operandos, OR para ponerlos a 1, y XOR para complementarlos. |
| Poner a 0 el acarreo (Clear Carry) | CLRC | Estas instrucciones sobre el bit de acarreo pueden definirse para los demás bits de estado del procesador. |
| Poner a 1 el acarreo (Set Carry) | SETC | |
| Complementar el acarreo | COMC | |

## INSTRUCCIONES DE DESPLAZAMIENTO

Permiten desplazar o rotar los bits de un operando de diversas formas:

| Nombre | Nemónico | Descripción |
|---|---|---|
| Desplazamiento lógico a la derecha | SHR |  |
| Desplazamiento lógico a la izquierda | SHL |  |
| Desplazamiento aritmético a la derecha | SHRA |  |
| Desplazamiento aritmético a la izquierda | SHLA |  |
| Rotación a la derecha | ROR |  |
| Rotación a la izquierda | ROL |  |
| Rotación a la derecha con acarreo | RORC |  |
| Rotación a la izquierda con acarreo | ROLC |  |

Algunos procesadores tienen un formato para la instrucción de desplazamiento con varios campos:

1º) OP: Código de operación que indica un desplazamiento.
2º) REG: Dirección que especifica la localización del operando.
3º) TYPE: Código de dos bits que indica el tipo de desplazamiento.
4º) RL: Código de un bit que indica la dirección de desplazamiento.
5º) COUNT: Código de k bits que indica el número de posiciones a desplazar.

# 9.- Instrucciones de control de programa.

Las instrucciones de un programa se almacenan en posiciones de memoria consecutivas. Cada vez que se extrae una instrucción de la memoria, el PC se incrementa para apuntar a la siguiente instrucción del programa.

Las instrucciones de control de programa permiten alterar el flujo de ejecución de los programas actuando sobre el contenido del PC. Con ellas, los procesadores pueden ejecutar diferentes secuencias de instrucciones en función de los resultados de cálculos previos.

| Nombre | Nemónico | Descripción |
|---|---|---|
| Bifurcación (*Branch*)<br>Salto (*Jump*) | BR<br>JMP | Normalmente se diferencian sólo en el modo de direccionamiento, que suele ser directo o indirecto en los saltos, y relativo en las bifurcaciones. Son instrucciones de una sola dirección, y transfieren la dirección efectiva al PC. Pueden ser incondicionales o condicionales. |
| Salto implícito (*Skip*) | SKP | Salta la siguiente instrucción de la secuencia si se cumple una determinada condición. Para ello incrementa el PC en su fase de ejecución. No necesita campo de direcciones. Puede combinarse con una instrucción de salto incondicional para hacer una bifurcación condicional. |
| Llamada a subrutina<br>Retorno de subrutina | CALL<br>RET | Instrucciones para el manejo de subrutinas. |
| Comparación | CMP | Realiza la comparación de dos operandos mediante una substracción, pero sin almacenar el valor de la resta. Puede producir un salto condicional, cambiar el contenido de un registro o modificar los bits de estado. |
| Test mediante AND | TEST | Realiza la función AND de dos operandos, pero sin almacenar el resultado. Puede producir los mismos efectos que la instrucción de comparación. |

## INSTRUCCIONES DE SALTO CONDICIONAL

Comprueban los bits de estado para una determinada condición: si es cierta, el control de programa se transfiere a la dirección efectiva, y si es falsa se continúa con la ejecución de la instrucción siguiente.

| Condición de bifurcación | Nemónico | Test | Descripción |
|---|---|---|---|
| Bifurcación si es cero | BZ | Z = 1 | El bit de estado Z se usa para comprobar si el resultado de una operación de la ALU es 0. |
| Bifurcación si no es cero | BNZ | Z = 0 | |
| Bifurcación si hay acarreo | BC | C = 1 | El bit de acarreo C se usa para comprobar el acarreo después de una suma o resta en la ALU. También se utiliza en las instrucciones de desplazamiento para comprobar el bit saliente. |
| Bifurcación si no hay acarreo | BNC | C = 0 | |
| Bifurcación si negativo | BN | N = 1 | El bit de estado N refleja el estado del bit más significativo de la salida de la ALU, tanto si representa el signo como si no. |
| Bifurcación si positivo | BNN | N = 0 | |
| Bifurcación si hay desbordamiento | BV | V = 1 | El bit de desbordamiento V se usa en operaciones aritméticas de números con signo. |
| Bifurcación si no hay desbordamiento | BNV | V = 0 | |

## INSTRUCCIONES DE SALTO CONDICIONAL

Las comparaciones de dos palabras A y B se hacen mediante una operación de resta (A-B). El resultado no se almacena, pero los bits de estado se ven afectados.

Para números binarios sin signo:

| Condición de bifurcación | Nemónico | Condición | Bits de estado |
|---|---|---|---|
| Bifurcación si es mayor | BH | A > B | C+Z = 0 |
| Bifurcación si es mayor o igual | BHE | A ≥ B | C = 0 |
| Bifurcación si es menor | BL | A < B | C = 1 |
| Bifurcación si es menor o igual | BLE | A ≤ B | C+Z = 1 |
| Bifurcación si es igual | BE | A = B | Z = 1 |
| Bifurcación si no es igual | BNE | A ≠ B | Z = 0 |

Para números binarios con signo:

| Condición de bifurcación | Nemónico | Condición | Bits de estado |
|---|---|---|---|
| Bifurcación si es mayor | BG | A > B | (N⊕V)+Z = 0 |
| Bifurcación si es mayor o igual | BGE | A ≥ B | N⊕V = 0 |
| Bifurcación si es menor | BL | A < B | N⊕V = 1 |
| Bifurcación si es menor o igual | BLE | A ≤ B | (N⊕V)+Z = 1 |
| Bifurcación si es igual | BE | A = B | Z = 1 |
| Bifurcación si no es igual | BNE | A ≠ B | Z = 0 |

**SUBRUTINAS**

Es relativamente frecuente que conjuntos de instrucciones en un programa se repitan. En lugar de copiar varias veces en memoria las mismas secuencias de instrucciones, se almacenan una única vez en lo que se denomina subrutina. Cada vez que se necesite ejecutar esa secuencia se hace un salto a la primera instrucción de la subrutina. Cuando se termine de ejecutar la subrutina, se hace otro salto para volver al programa principal.

La instrucción de llamada a subrutina CALL posee un campo de dirección y realiza dos operaciones:

1. Guarda el valor del PC en una zona de almacenamiento temporal (dirección de retorno).
2. La dirección de la instrucción CALL se carga en el PC (primera instrucción de la subrutina).

La última instrucción de una subrutina es la de retorno al programa principal RET: carga en el PC la dirección almacenada por la instrucción CALL.

Algunos procesadores utilizan una pila como almacenamiento temporal de las direcciones de retorno, lo cual permite manejar subrutinas anidadas:

| | | |
|---|---|---|
| CALL dir: | SP ← SP-1 | Decrementa el puntero de pila. |
| | M[SP] ← PC | Almacena la dirección de retorno en la pila. |
| | PC ← dir | Transfiere el control a la subrutina. |
| RET: | PC ← M[SP] | Transfiere la dirección de retorno al PC. |
| | SP ← SP+1 | Incrementa el puntero de pila. |

# 10. Arquitectura de procesadores

El procesador es el coordinador de todas las tareas que realiza un PC, esto significa su diseño interno es muy importante. En su interior, no es más que un conjunto de bloques interconectados entre sí en el cual cada uno de ellos realiza una función. El diseño de esos elementos y cómo se interconectan es lo que se llama arquitectura del procesador.

Para funcionar, una computadora lee instrucciones y datos. La velocidad a la que lee datos y realiza cálculos, viene determinada por la famosa frecuencia de funcionamiento que puedes ver en cualquier folleto de un micro y que se mide en gigahertzios. Ese reloj sirve para coordinar a todo el sistema así que en principio un procesador a 2 GHz puede hacer el doble de trabajo en el mismo tiempo que uno de 1 GHz pero no es tan sencillo. Gracias a los avances producidos en el diseño de la arquitectura, las prestaciones cada vez dependen menos de esa frecuencia de funcionamiento. Puede ocurrir, por tanto, que un micro con una menor velocidad sea capaz de realizar más tareas.

Para entenderlo, hay que revisar la historia. Cuando se crearon los primeros microprocesadores, estos sólo eran capaces de realizar una operación en cada ciclo de reloj y algunas, las más complejas, podían incluso llevar más tiempo. Sin embargo, gracias a cambios arquitectónicos, cualquier procesador actual es capaz de procesar varias instrucciones al mismo tiempo. Por supuesto, cada arquitectura es distinta lo cual lleva a que dos procesadores funcionando a la misma velocidad no tengan que funcionar igual de rápido.

Además, cuando se cambia la arquitectura, se pueden añadir más bloques que ejecuten nuevas instrucciones. Es lo que ocurre con los conjuntos de instrucciones SSE y AVX, por ejemplo. Estas, permiten acelerar ciertos cálculos asociados a programas matemáticos, científicos, financieros y de seguridad. Al incluir estos bloques, se consigue aumentar la velocidad de ejecución de forma radical, en algunas utilidades se puede realizar el doble o incluso más de trabajo en el mismo tiempo, pero se necesita que los desarrolladores de software vuelvan a crear los programas de nuevo.

El proceso es continuo, mejoras en las técnicas de fabricación permiten hacer transistores más pequeños que dejan más espacio para añadir más funcionalidades. La primera opción que se tomó fue la más lógica. Se añaden más núcleos, que no es más que replicar procesadores simplificados e interconectarlos entre sí. Tras esto, se tiende a integrar cada vez más elementos dentro de la CPU. Digamos que los elementos pasan de la placa base poco a poco al micro. Se van integrando por ejemplo, el controlador de memoria, o la tarjeta gráfica.

A tanto ha llegado esta evolución, que se empieza a utilizar el concepto de APU en vez del de CPU. En un futuro no muy lejano se tenderá a SOC (system on a chip) es decir a que todo el sistema se encuentre en el interior de una pastilla.

Aparte de la parte lógica, no debemos de olvidar que un micro es un dispositivo físico. Como tal, está limitado en aspectos tales como la temperatura o el voltaje. En esencia, la potencia consumida por un procesador es proporcional a la frecuencia de funcionamiento. Es decir, si aumentamos en un 30% la velocidad también la potencia aumentará en la misma proporción. Si el sistema de refrigeración, normalmente un ventilador, no es capaz de disipar esa potencia, la temperatura aumentará de manera constante hasta que el dispositivo se queme o se pare.

Los fabricantes, teniendo en cuenta esto, añaden sistemas a los procesadores para acelerar su funcionamiento siempre y cuando estemos dentro de unos límites físicos aceptables. Ejemplos de tecnologías que realizan esto son los famosos Turbo Core de AMD y Turbo Boost de Intel. El procesador puede funcionar más lento en caso de que el sistema no necesite tanta velocidad de ejecución.

Por supuesto, la arquitectura también define cómo se comporta el procesador en relación a otros elementos de la placa base. Por ejemplo la conexión con las memorias o los canales de acceso a las tarjetas gráficas.

En definitiva, la arquitectura es junto a la tecnología de fabricación lo que define las características y las posibles prestaciones de un microprocesador. Como ves, todo esto hace que sea cada vez más complejo la comparación entre los distintos procesadores ya que puede ser que algunos sean más rápidos que otros en un tipo de aplicaciones y en otras no

## Historia

De los primeros procesadores y arquitecturas hemos pasado a equipos móviles que aúnan más potencia de la que hubiéramos pensado hace poco. Te invitamos a dar un paseo por la historia, el desarrollo y las claves que debes conocer para entender el mundo de los procesadores y su arquitectura.

## El origen de los procesadores

En la prehistoria de la informática sólo unos pocos cientos de personas en todo el mundo tenían las aptitudes necesarias para ponerse a los mandos de los ordenadores de la época. Allá por los años 50 y 60, los computadores eran máquinas que ocupaban varias habitaciones e incluso necesitaban de múltiples operarios para funcionar correctamente. Apenas trabajaban para grandes universidades o ejércitos y eran completamente desconocidas en los hogares.
Es evidente que esto ha cambiado radicalmente con el paso de los años y ahora cualquiera tiene a su disposición un ordenador sencillo de utilizar. Pero antes no era así. Entonces, ¿cómo empezó a fraguarse la era de la Información?

### Las máquinas que cambiaron el mundo

Tras la era industrial y aprovechando la segunda guerra mundial, los gobiernos más potentes se vieron en la necesidad de invertir el tiempo de sus mejores científicos en idear máquinas que pudiesen tratar información, ya sea procedente del enemigo y en ese caso descifrándola, o enviándola oculta a un aliado. Para ello necesitaban disponer de una estructura y organización que les permitiese procesar información que estuviese en un formato conocido, y además almacenar los resultados para poder estudiarlos a posteriori. Nació la arquitectura de Von Neumann

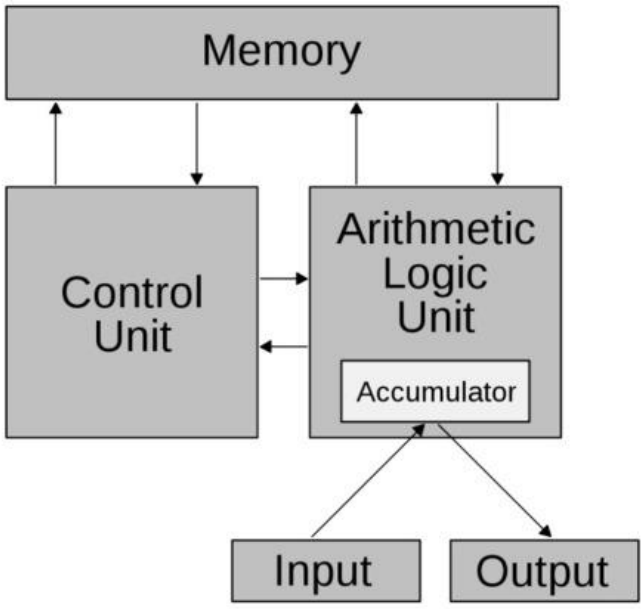
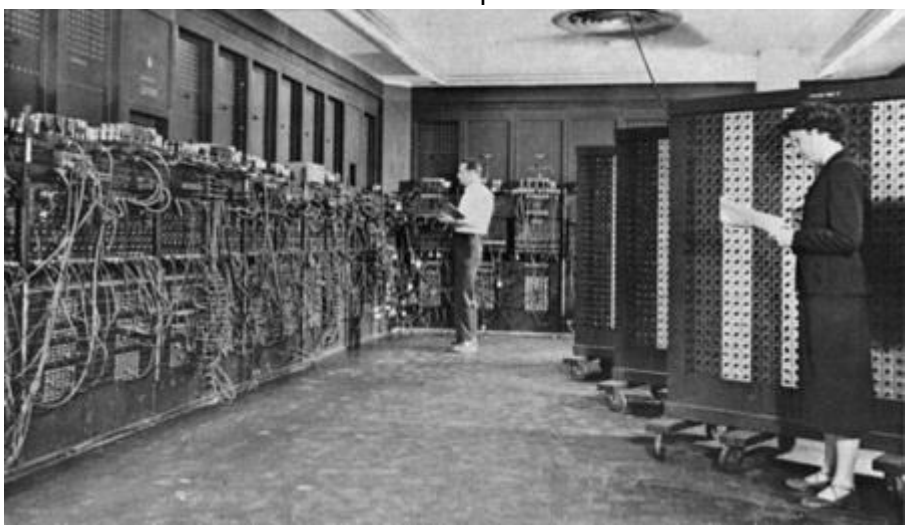


Diagrama de la arquitectura de Von Neumann

La arquitectura de Von Neumann fue ideada por John Von Neumann en la década de los 40 para un trabajo del gobierno americano, y hoy en día sigue vigente en la mayoría de los computadores modernos. Su misión es servir un esquema de funcionamiento genérico que permitiese a un ordenador ejecutar tareas sucesivamente, sin tener que modificar la estructura física del equipo. Según esta organización, el computador dispone de una lista de instrucciones que un procesador interno va descifrando y más tarde ejecutando. Una vez que ha finalizado esa ejecución se pasa al siguiente elemento de la lista, y se repite el proceso. Para ello se necesitan tres componentes básicos:

- Una unidad de proceso que incluirá ALU y unidad de control.
- Dispositivos de memoria para almacenar información, bien temporal o final.
- Periféricos de entrada/salida para comunicarse con el usuario.



El "Electronic Numerical Integrator And Computer" del ejército estadounidense

La arquitectura de Von Neumann fue utilizada por primera vez en el ENIAC o Electronic Numerical Integrator And Computer, que trabajó para el gobierno estadounidense entre los años 1946 y 1955. Sobre él hablaremos en el último apartado.

¿Y cómo trabajarían estos primeros computadores? Mediante el sistema binario. Se trata de un sistema de numeración con un funcionamiento similar al decimal al que estamos acostumbrados los seres humanos, pero utilizando únicamente el 0 y el 1. A partir de estos dos números se construye toda la información, si bien es cierto que trabajar en binario es algo bastante poco habitual al existir otras herramientas mucho más avanzadas y fáciles de interpretar por el ser humano. Lo mejor del sistema binario es que es un sistema simple que permite construir circuitos electrónicos para realizar operaciones matemáticas de una forma sencilla a través de circuitos elementales.

El procesador dispone de una serie de circuitos electrónicos que son utilizados por los algoritmos, ideados por el ser humano para afrontar problemas. ¿Qué es, entonces, un algoritmo?

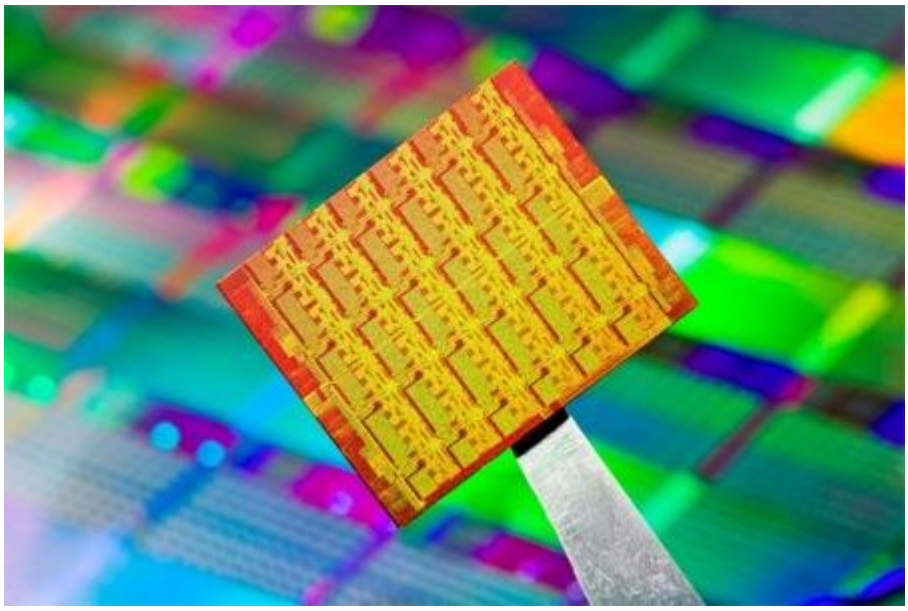
- Paso 1: Poner la sartén en la vitrocerámica
- Paso 2: Echar aceite
- Paso 3: Calentar el aceite
- Paso 4: Esperar a que esté caliente
- Paso 5: Cascar el huevo
- Paso 6: Verterlo con cuidado sobre el aceite caliente
- Paso 7: Con la ayuda de una paleta, echar el aceite por encima del huevo
- Paso 8: Comprobar que el huevo ya está cocinado y, en ese caso, sacarlo a un plato

Un algoritmo es una secuencia de órdenes o instrucciones que se dictan en un cierto orden. Es necesario que cada paso esté muy bien definido y que se siga un orden estricto para que la máquina sea capaz de

ejecutarlo sin problemas. El ejemplo de arriba es un símil respecto de la vida real, el algoritmo de cómo elaborar un huevo frito con puntilla. A continuación os dejo con otro ejemplo:

- Paso 1: coge el valor del registro A y llámalo N1.
- Paso 2: coge el valor del registro B y llámalo N2.
- Paso 3: suma N1 y N2 y almacena el resultado en el registro C.
- Paso 4: almacena el valor del registro C en la posición de memoria etiquetada como M.

Con los cuatro pasos de este segundo ejemplo nos hemos aproximado mejor a la realidad de nuestros ordenadores: hemos sumado dos números (existentes en los registros A y B) y su resultado lo hemos almacenado en la posición de memoria M. Podríamos incluso decir que hemos utilizado un pseudolenguaje ensamblador, y recuerdo que toda la información que maneja el ordenador está basada en 0 y 1, incluyendo no sólo los valores numéricos de la propia suma si no también las direcciones en las que se almacena la información (en este caso, las direcciones de los registros A, B, C y también la posición de memoria etiquetada como M).



Procesador prototipo de Intel de 2009, con 48 núcleos

En definitiva, el procesador se encarga de recibir secuencias de órdenes y ejecutarlas. Estas órdenes serán mayoritariamente matemáticas (suma estos dos números y guarda el resultado en esta determinada posición de memoria) pero también de almacenamiento o interrupciones del sistema. Y precisamente estas órdenes simples y atómicas se denominan instucciones, que son las operaciones que un procesador es capaz de entender y ejecutar. Por ejemplo, suma dos números y almacena el resultado en esta memoria, o multiplica estos dos números, o algo mucho más simple como almacena este dato en esta posición de la memoria. Las instrucciones son operaciones muy simples pero con las que se construye todo, y un conjunto de estas instrucciones se denomina set de instrucciones o ISA (Instruction Set Architecture). Por ejemplo, x86 es la ISA de los procesadores Intel o AMD domésticos actuales, los cuales a su vez utilizan múltiples microarquitecturas, y ARM es la ISA de los procesadores de Samsung, Qualcomm, Apple, etc. En posteriores artículos entraremos más a fondo en los diferentes juegos de instrucciones así como en las arquitecturas más representativas del mercado actual.

## ¿Y qué es una microarquitectura?

Hablar de procesadores es una introducción esencial para continuar hablando de microarquitectura. ¿Qué es una microarquitectura?

Un microprocesador no es sólo un cerebro que procesa información (técnicamente la parte que realiza las operaciones se llama ALU, Arithmetic Logic Unit o unidad aritmético lógica), sino mucho más. Está compuesto de registros (pequeñas memorias donde se almacenan datos), buffers, cachés, unidades de proceso, ALU, y mucho más. Todo esto se fabrica utilizando componentes electrónicos ciertamente pequeños (las arquitecturas actuales de nuestros ordenadores utilizan transistores de 22 nanómetros, 0.000022 milímetros) y no siempre nos encontraremos con todos. Es necesaria una organización y estructuración de todos los componentes a la que se denomina microarquitectura.

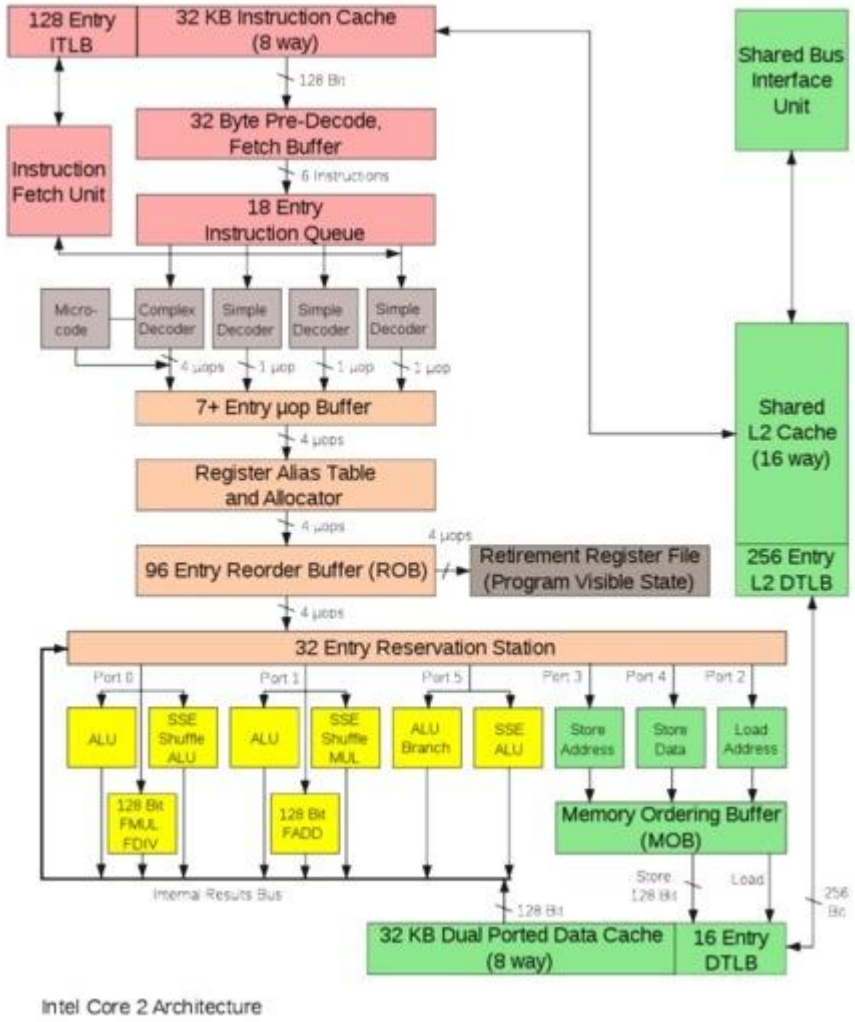



Diagrama de la arquitectura de un Intel Core 2 Duo

La microarquitectura pretende organizar todos los componentes internos de un procesador. Lo que tenéis aquí arriba es un diagrama de bloques de lo que nos encontraremos dentro del procesador, en este caso concreto de un Intel Core 2 Duo. Por ejemplo, la zona amarilla es la encargada de realizar las operaciones y la verde de almacenar las siguientes instrucciones a ser ejecutadas. Como véis es algo bastante complejo, y eso que no es más que un esquema muy general.

Uno de los aspectos más interesantes es que un mismo juego de instrucciones (ISA) puede ser ejecutado sobre diferentes microarquitecturas, que es lo que da lugar a las grandes rivalidades que por ejemplo tienen Intel y AMD bajo x86, o Samsung, Apple y Qualcomm con ARM. Sobre el estado actual del mercado y las grandes guerras en lo referente a las arquitecturas hablaremos en posteriores artículos.

## Los primeros procesadores

Los primeros procesadores no funcionaban con transistores si no con válvulas de vacío, y fue la Segunda Guerra Mundial la que propició que los gobiernos investigasen en máquinas que fuesen capaces de operar con información de forma muy rápida en comparación con la velocidad de cálculo humana. Uno de los primeros equipos era el ENIAC, el primero que siguió la arquitectura de Von Neumann y sobre el cual hay unos datos bastantes interesantes en su entrada de Wikipedia:

- Contenía 17.488 tubos de vacío, 70.000 resistencias y 10.000 condensadores, entre otros muchos componentes.
- Su peso total era de 27 toneladas, ocupando una superficie de 167 metros cuadrados.
- Para hacerla funcionar era necesario operar 6000 interruptores... manualmente, claro.
- Requería una potencia de 160 kiloVatios.
- Permitía operar hasta 5.000 sumas y 357 multiplicaciones por segundo.

Empezó a operar en 1946 tras casi tres años de diseño, desarrollo y fabricación por parte de J. Mauchly y J. Presper Eckert en la Universidad de Pennsylvania. Cesó su labor el 2 de octubre de 1955.

Si bien esta primera generación de computadores se caracterizó por trabajar con válvulas de vacío, programarse en código máquina o, a lo sumo, ensamblador, la segunda generación de procesadores evolucionó de forma muy notable debido a la escasa fiabilidad y durabilidad de las primeras propias válvulas. Empezó la carrera de los transistores, el almacenamiento magnético (es decir, los primeros discos duros), los lenguajes de alto nivel y los primeros sistemas operativos monousuario. Por entonces IBM ya llevaba varios años dando guerra y puso en el mercado una de las máquinas más avanzadas y exclusivas de la historia en su contexto, el IBM 7090 que es considerado como el primer ordenador con CPU de transistores de la historia. Era seis veces más potente que su antecesor (el IBM 709) y costaba la mitad, así que era una inversión asegurada.

Lo que hay frente a la silla de la derecha es un 7090

Esto es solamente el inicio de los tiempos, la prehistoria de los ordenadores. Y las cosas han cambiado mucho desde entonces. En la próxima entrada hablaremos de los diseños RISC y CISC, las repercusiones que han tenido en los dispositivos más modernos y también sus microarquitecturas más importantes, que con toda seguridad las estáis utilizando en este preciso instante.

## CISC frente a RISC

Habiendo tratado ya parte el inicio de la computación, hoy entraremos en una época algo más moderna. Muchos la considerarán la época dorada de la computación, aquella en la que germinó el concepto de los ordenadores actuales y se empezó a fraguar la era más contemporánea de la información.
Situémonos en los últimos años de la década de los 50: los procesadores se desarrollaban de forma completamente independiente, haciendo que un mismo programa tuviese que ser modificado para ejecutarse en máquinas diferentes. Es evidente que era una situación que había que cambiar.

### La operación más sencilla para un procesador

En la primera entrada hablamos de la ISA (Instruction Set Architecture) y pusimos dos ejemplos: ARM y x86 son las dos ISA más utilizadas en la actualidad en los procesadores de Intel, Qualcomm, ARM, Samsung, etc.
Una ISA esta formada por múltiples componentes, tales como los tipos de datos que maneja un procesador, los registros y sus tamaños, los buffers e incluso los errores que es capaz de manejar. Uno de los componentes fundamentales de cada ISA son el conjunto de instrucciones que admite.
Estas instrucciones son fundamentales e imprescindibles, ya que es lo que el procesador ejecuta. Dependiendo de la ISA existen múltiples tipos de instrucciones, aunque principalmente se engloban dentro de tres grandes categorías: operaciones con memoria, operaciones aritméticas y operaciones de control sobre la CPU.
Llegados a este punto, la complejidad del set de instrucciones es variable y depende enormemente de lo que sus diseñadores decidieran en el día de su creación. En nuestro ejemplo del algoritmo real del huevo frito recordaréis que teníamos un Paso 6: Verterlo con cuidado sobre el aceite caliente, que es una de sus instrucciones. Si realizamos una aproximación desde el punto de vista de un procesador esa sería una instrucción compleja que podría dividirse en varias instrucciones más simples:

- Paso 6.1: Colocar el huevo partido sobre la sartén.
- Paso 6.2: Acercar el huevo partido a un par de centímetros del aceite caliente.
- Paso 6.3: Mover verticalmente el huevo partido.
- Paso 6.4: Verter el contenido del huevo partido sobre el aceite hasta que esté vacío.
- Paso 6.5: Retirar el huevo partido y ya vacío.
- Paso 6.6: Tirar a la basura el huevo partido y ya vacío.

Como veis estamos realizando la misma labor con dos juegos de instrucciones diferentes. La primera instrucción Paso 6 es más compleja que las nuevas instrucciones Paso 6.X, que son más simples y sencillas de entender pero también son muchas más en cantidad. Elegir entre una u otra dependerá del set de instrucciones que admite el procesador.
Este razonamiento es la base para entender lo que ocurrió con CISC a mediados del siglo XX, cuando IBM se propuso unificar las instrucciones con las que trabajaban los procesadores. Unos años más tarde y teniendo en cuenta su experiencia, introdujo con otro enfoque: RISC. Ambas aproximaciones se fundamentan sobre la misma base de funcionamiento (un bucle infinito en el que en un mismo ciclo, un procesador recibe una nueva instrucción a ejecutar, la decodifica, la ejecuta y espera a la siguiente instrucción) pero son muy diferentes en cuanto al tipo de instrucción que admiten.

## CISC acude en la búsqueda de lo más completo

En los años 50, todos los computadores se diseñaban de forma completamente aislada unos de otros. Esto hacía que sus instrucciones fuesen independientes, haciendo que un programa escrito para un cierto ordenador no se pudiese ejecutar en otro. A finales de la década, IBM reunió a un grupo de sus

investigadores para estudiar la forma con la que un programa pudiese trabajar en múltiples computadores sin importantes cambios, ampliando la compatibilidad del software en diferentes máquinas. El resultado fue el enfoque CISC, Complex Instruction Set Computing, introducido por primera vez en los IBM System/360 el 7 de abril de 1964.

Un System/360 en una fábrica de Volkswagen, año 1973

CISC ofrece un conjunto de instrucciones bastante completas y lentas de ejecutar, pero que agrupaban varias operaciones de bajo nivel en la misma instrucción. Esto da lugar a programas pequeños y sencillos de desarrollar que además realizaban pocos accesos a memoria: esto que ahora podría parecer insignificante era vital en aquella época, cuando los ordenadores trabajaban con muchos menos recursos que los equipos actuales. En el algoritmo de ejemplo del huevo frito, con un enfoque CISC tendríamos una única instrucción: nuestro Paso 6: Verterlo con cuidado sobre el aceite caliente.

En la actualidad CISC tiene a x86 como su mayor exponente, con AMD y sobre todo Intel a la cabeza de su desarrollo. Hay muchos ejemplos históricos como los PDP, Motorola 68000, Intel 4004 o Intel 8086, quizá los más representativos. Prácticamente cualquier ordenador de sobremesa o portátil desde los años 80 ha utilizado un procesador x86.

### RISC, con la simpleza por bandera

Tras el lanzamiento de CISC, los científicos de IBM empezaron a comprobar que los diseñadores de software creaban sus propias instrucciones más simples y precisas. Entonces, ya en la década de los 70, empezaron a diseñar una alternativa que posteriormente se introdujo en el mercado bajo el acrónimo RISC, Reduced Instruction Set Computing. El IBM 801 que empezó a crearse en 1975, fue diseñado por John Cocke y es considerado el primer procesador RISC de la historia.

John Cocke ya en los 80 junto a uno de los prototipos que usó el procesador 801

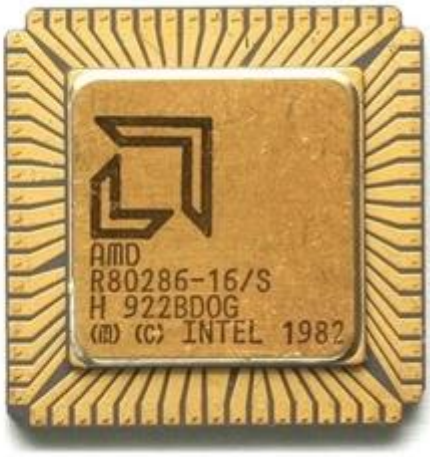
La principal virtud de RISC es tener un conjunto de instrucciones muy simples que se ejecutarán más rápidamente en el procesador. Existe un catálogo de pocas instrucciones y éstas son muy sencillas, lo cual implica también que para una cierta tarea compleja necesitaremos un mayor número de ellas, y por esto el programa final tendrá una longitud mayor y además accederá en un mayor número de ocasiones a los datos almacenados en la memoria. En nuestro ejemplo del algoritmo del huevo frito, un procesador RISC estaría compuesto por las instrucciones descritas entre Paso 6.1 y Paso 6.6.

Un procesador de tipo RISC es más simple tanto en software (instrucciones) como en hardware (registros de memoria), lo cual hace que sea un dispositivo notablemente más barato que otras CPU. En la actualidad el mayor ejemplo de procesador RISC son los productos ARM, utilizados ampliamente en dispositivos móviles pero también en otros campos como los supercomputadores. ARM será el principal protagonista de una de nuestras próximas entradas de esta saga.

## RISC frente a CISC y la gran batalla actual

CISC nació con la finalidad de homogeneizar los diferentes computadores en los años 50 y 60. RISC buscó en los 70 ir un paso más allá y mejorar el rendimiento con instrucciones más simples pero programas más largos y más difíciles de desarrollar.

Tanto CISC como RISC han evolucionado de forma muy notable desde su nacimiento, adoptando mejoras provenientes del contrario en ambos casos y nuevos conjuntos de instrucciones para adaptarse a los usos de los ordenadores. Si bien es cierto que en la época de su creación la diferencia era muy amplia -principalmente debido a las limitaciones técnicas de la época tanto en tamaño de memoria como en velocidad de proceso -, en la actual informática moderna los requisitos son muy diferentes. Los límites en capacidad de almacenamiento son casi inexistentes y los procesadores son capaces de ejecutar millones de instrucciones en un solo segundo.


¿Adivinas qué enfoque siguen estos dos?
La gran batalla actual es la de sus dos grandes exponentes, ARM y x86, que han actualizado sus objetivos a lo que les importa a los usuarios del siglo XXI. Veremos que el punto fuerte de ARM está en la eficiencia energética. Un chip ARM consume mucha menos energía que un procesador x86 que tiene en su alto rendimiento su gran virtud, a costa de consumir bastante más energía.
Ésta es la gran batalla que nos ocupará las próximas entradas.

# ARM

Hablaremos de ARM, una arquitectura pero también una empresa diseñadora con una dilatada experiencia y una interesante historia que empieza en Reino Unido en los años setenta.
ARM es una arquitectura que ha tenido un enorme crecimiento en los últimos años, si bien su nacimiento se remonta tres décadas en el pasado de la mano de Acorn Computers Ltd., una empresa ya extinta, y con quien colaboran compañías históricas de gran importancia en el mercado actual.

## Un poco de historia

En 1980, la BBC (British Broadcasting Corporation) se interesó en crear la Computer Literacy Project, una serie educativa en la que querían proponer ejercicios y tareas de programación, sonido o gráficos que deberían completarse con un computador. Para ello inició un concurso mediante el cual buscaban una empresa británica que les diseñase un equipo a medida, con una serie de requisitos tales como un bajo precio pero a la vez múltiples funcionalidades y posibilidades.
Acorn Computers Ltd., fundada por Hermann Hauser y Chris Curry y afincada en Cambridge, ya había presentado en marzo de 1979 el Acorn System 1 tan solo un año después de su fundación, y vio en el concurso de la BBC una interesante oportunidad. A principios de 1980 presentaron al concurso el Acorn Proton.


Acorn Atom, 1980

Se trataba de una evolución respecto de su predecesor, el Acorn Atom: un ordenador destinado al mercado doméstico y puesto a la venta en 1980, con un procesador MOS 6502 a 1 MHz., 2 KB de memoria RAM y 8 de ROM, en ambos casos ampliables hasta los 12 KB. Incluso disponía de periféricos como lector de cintas, el almacenamiento más común de la época. En el Acorn Proton también estábamos ante un procesador MOS 6502 pero a 2 MHz., además de mayor cantidad de memoria RAM (16 KB) y un mayor número de periféricos asociados, como acceso a cintas, redes o incluso impresora.

El Acorn Proton luchó contra otros diseños de compañías como Sinclair o Dragon y salió vencedor por a su alta potencia - especialmente en el apartado gráfico - y reducido precio: unas 235 libras. En abril de 1981 la BBC llegó a un acuerdo con Acorn para que fabricasen 12.000 unidades del Proton de cara a la primavera de 1982, cuando el Computer Literacy Projectiniciaría su andadura en televisión. Fue entonces cuando se modificó el nombre y el Acorn Proton pasó a denominarse BBC Micro.

British Broadcasting Corporation Micro, 1982, con enorme parecido al Atom

La BBC supo mover su proyecto y el BBC Micro fue un completo éxito. Acorn vendió 24.000 unidades en 1982 que ascendieron hasta el millón en toda la vida del Micro, desde su lanzamiento hasta el año 1994 en el que se retiró del mercado. Éxito para la BBC y mucho dinero para Acorn, una compañía que a pesar de tener unos pocos años de vida logró introducirse en el mercado de los computadores domésticos de una forma realmente veloz.

Tras el BBC Micro, Acorn empezó a plantear nuevos desarrollos en los años posteriores. Diseñó nuevo hardware para competir contra los equipos domésticos de la época de compañías de renombre como Amstrad, Sinclair o Apple, y a la vez intentó entrar en el mercado profesional con chips de alto rendimiento basados en el enfoque RISC.

En 1983, Acorn inició el diseño de lo que un par de años más tarde fue Acorn RISC Machine, ARM, y que tuvo en el ARM1 su primer chip final pero aún no comercial. Acorn mandó su fabricación a VLSI Technology, Inc., que lo fabricó a partir del 26 de abril de 1985 en pocas unidades que fueron únicamente utilizadas por la propia Acorn de forma interna, para diseñar una segunda versión ya pública: ARM2.

ARM2 era el núcleo de un coprocesador matemático de apoyo a la CPU que empezó a utilizarse en el BBC Micro, gracias a su alta escalabilidad, y que posteriormente también llegó al mercado con una versión compatible con los IBM PC de la época. Ofrecía un rendimiento de unos 4 MIPS (millones de instrucciones por segundo) funcionando a 8 MHz., con un consumo de 0.1 vatios y el uso de unos 30.000 transistores (por unos 68.000 del competidor Motorola 68000).

En este período en el que se desarrollaron los ARM1 y ARM2, Acorn tuvo serios problemas económicos. En 1983 presentó el Acorn Electron, una versión mejorada del BBC Micro que compitió contra los míticos Commodore 64 y Sinclair ZX Spectrum de la época. Sufrieron problemas de fabricación del equipo (ordenaron 300.000 unidades pero el fabricante sólo pudo crear 30.000) que llevaron a los usuarios a adquirir los productos de Commodore y Sinclair en una campaña tan importante como la de la Navidad, en este caso del año 1984. El stock llegó unos meses más tarde y Acorn acumuló 250.000 unidades del Electron sin vender. La herida estaba hecha e incluso Acorn llegó a un acuerdo con Olivetti para que comprase algunas participaciones.

A pesar de ello, Acorn continuó fabricando nuevos ordenadores de uso tanto doméstico como profesional, como lo fueron los BBC Master (1986) y Acorn Archimedes (1987). Se trata de dos equipos que utilizaron la tecnología ARM, el primero en forma de coprocesador y el segundo como procesador central, con precios bastante altos (499 y desde 800 libras, respectivamente) y que no funcionaron del todo mal en el mercado, principalmente en el entorno académico (colegios y universidades). El Acorn Archimedes es especialmente importante ya que fue el primer ordenador doméstico del mercado en utilizar la arquitectura ARM y un sistema operativo propio: Arthur OS, desarrollado por Castle Technology Limited y que más adelante pasó llamarse ARM OS.

Acorn Archimedes, 1987

## El nacimiento de ARM junto a Apple y el fin de Acorn Computers

En la recta final de los ochenta Acorn se encontraba evolucionando su tecnología, presentando las arquitecturas ARMv2 y preparando la ARMv3. Fue entonces cuando Apple se interesó en colaborar con Acorn para diseñar un nuevo procesador para un dispositivo que se presentaría unos años más tarde, en 1993: la Apple Newton, el primer dispositivo en la historia en ser considerado PDA (personal digital assistant). En esos años terminó el diseño de la arquitectura ARMv3 y se presentó la familia de procesadores ARM6, compuesta por tres modelos: ARM60, ARM600 y ARM610, este último utilizado en la primera Newton MessagePad 100. Pero también algo muy importante: debido a la alta carga de trabajo, Acorn creó junto a Apple y VLSI Technology una compañía que se encargase exclusivamente del diseño de ARM. En noviembre de 1990 nació Advanced RISC Machines Ltd, que ya en el año 1998 con su salida se transformó en la actual ARM Holdings.

En sus primeros años de vida ARM creció de forma muy notable y licenció sus productos a múltiples compañías como Intel (que presentó los primeros XScale basados en la arquitectura ARMv5; el 27 de junio de 2006 vendió el negocio a Marvell) o IBM. En la actualidad son varias decenas las empresas que crean o han creado procesadores ARM, muchas de ellas de primera línea en el panorama tecnológico: las propias Intel, IBM o Apple (quien vendió parte de sus participaciones de ARM Ltd. en 1999 por unos 59 millones de dólares), pero también Broadcom, Huawei, LGDEC, Qualcomm, AMD, RIM, Samsung, Panasonic, Sony o Texas Instruments, entre muchos otros.

Por su parte, Acorn Computers inició a principios de los 90 un descenso en caída libre sin visos de detenerse. Tras modificar su estructura organizativa, Acorn quiso introducirse en el mercado de los primeros sistemas de televisión bajo demanda, continuó en el mercado educativo y mantuvo la colaboración con la BBC así como sus desarrollos de estaciones de trabajo para el mercado más profesional. Las cosas no funcionaron como en la anterior década, Olivetti vendió sus participaciones a Lehman Brothers en 1996 y todo se mantuvo con pérdidas cada vez mayores. En enero de 1999 cambiaron el nombre a Element 14 y en mayo iniciaron las conversaciones con MSDW Investment Holdings Limited, una filial de Morgan Stanley, para la venta de la compañía. El acuerdo finalizó en junio del mismo año, cuando se vendió Element 14 por 270 millones de libras.

Tras ello, Morgan Stanley jugó con Element 14 y terminó vendiéndola como una empresa de fabricación de DSL a Broadcom. Poco después, en noviembre de 2000, Element 14 pasó a formar parte del negocio de Broadcom como sucursal encargada del diseño de routers DSL, y tras ello desapareció del mercado.

## ARM, la arquitectura RISC más importante de la historia

Cuando en la anterior entrada hablábamos de RISC, mencionábamos que tenía a ARM como su principal representante en el mercado actual. Es cierto que existen otras arquitecturas con el mismo enfoque, como lo son los IBM PowerPC o las MIPS32 y MIPS64 de MIPS Technologies, pero su uso es mucho menor que en el caso de nuestra principal protagonista.

ARM nació como una arquitectura para uso en ordenadores y dispositivos de bolsillo, con aquella primera Newton de Apple como primer gran dispositivo en movilidad. En la actualidad se ha ampliado enormemente los dispositivos en los que se integra: se utilizan en teléfonos y tabletas, por supuesto, en reproductores y grabadores de vídeo (DVD, Blu-Ray, etc.), videoconsolas portátiles o incluso en modems y routers de comunicación. Pero también en televisores, frigoríficos, lavadoras o lavavajillas, en teléfonos DECT o en coches (por eso ahora incluyen tantas funciones). Los aspiradores robot e incluso juguetes como Lego Mindstorms NXT utilizan un procesador ARM.

Hay otro uso algo desconocido pero también bastante convencional: el uso de chips ARM en microcontroladores de gestión de otros dispositivos, como por ejemplo un disco duro tradicional o un SSD. En el segundo caso, cuando nos referimos al controlador incluído (por ejemplo SandForce) nos referimos a un chip que integra un procesador basado en ARM que es el encargado de gestionar las posiciones

libres y ocupadas de memoria, de ejecutar TRIM y, más en general, de organizar y administrar la estructura interna del almacenamiento.

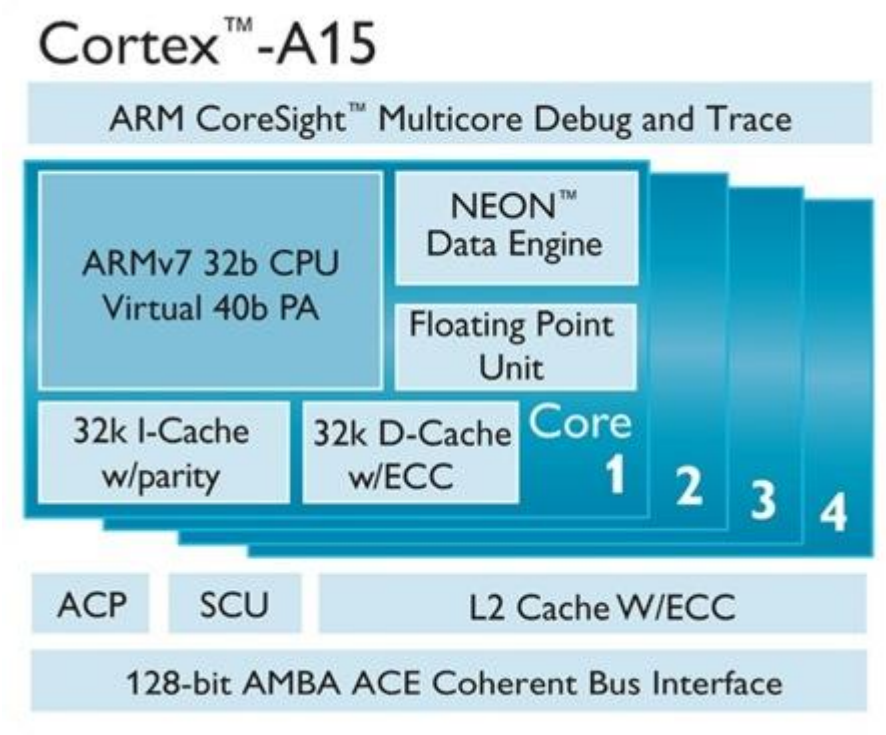

Controlador Indilinx Everest 2 de un Vertex 4 rodeado de chips de memoria

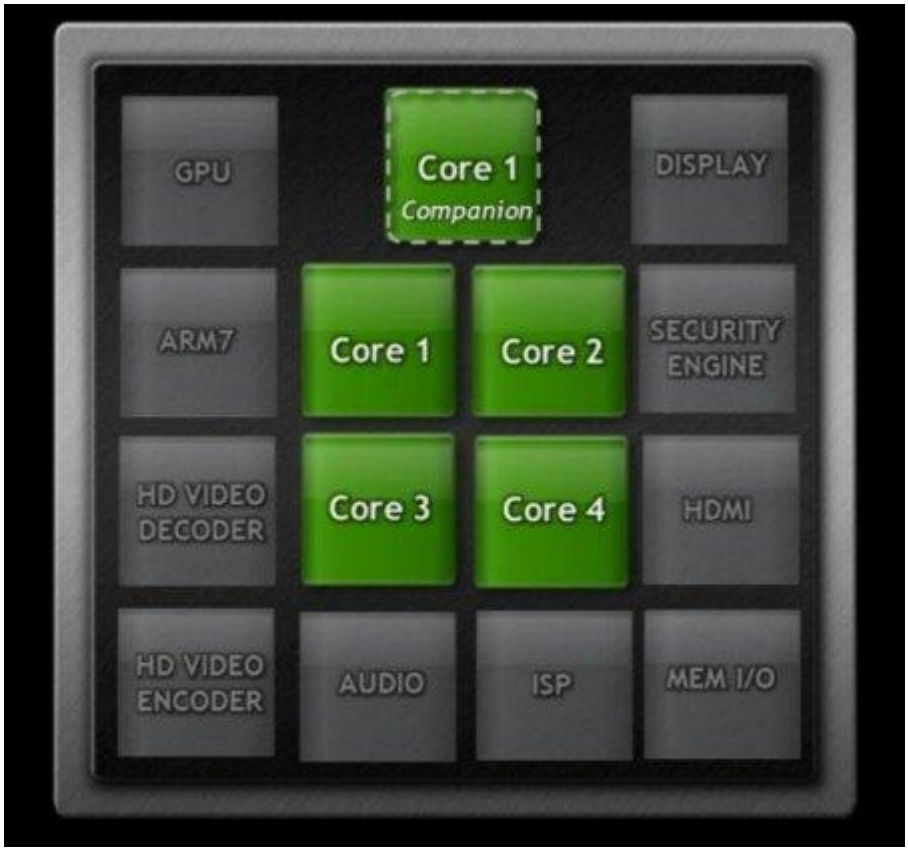
Todos estos usos de procesadores ARM existen porque es una arquitectura de muy bajo consumo energético, al menos en comparación con las alternativas existentes en el mercado. Muchas veces incluso podría pasar desapercibido respecto del consumo de otros componentes del mismo dispositivo, por ejemplo en un coche eléctrico. Además, la fabricación de estos chips es bastante barata, si bien es cierto que suele requerir de implementaciones software específicas que sí suelen ser más costosas que en otras arquitecturas como la x86 más habitual de nuestros ordenadores. Ella será la protagonista de nuestra próxima entrada de este especial.

## Lo que hace a ARM diferente

Desde la primera versión comercial allá por mediados de los 80, ARM ha ido evolucionando con el paso del tiempo y por supuesto se han introducido nuevas instrucciones con cada versión de la arquitectura, no sólo para mejorar el rendimiento si no también la seguridad. En Wikipedia hay una amplia lista que contempla las arquitecturas y los núcleos que han existido a lo largo de la historia de ARM, desde la primera ARMv1 (núcleo ARM1) hasta la actual ARMv7 (núcleos Cortex-A, Cortex-M y Cortex-R) o incluso la próxima arquitectura ARMv8 (con los Cortex-A50) que ya ha sido diseñada y cuyos primeros productos se esperan para el 2013.

También hay otros fabricantes que han creado sus arquitecturas basándose en las de ARM, en la actualidad por ejemplo Qualcomm (y sus ARMv7 Snapdragon, modelos Scorpion y Krait), NVidia (Tegra en todas sus vertientes), Apple (ARMv7 en los últimos A6 y A6x) o Samsung (y sus Exynos, hasta ahora todos basados también en ARMv7).




Una de las principales características de ARM es que utiliza relativamente pocos transistores, o al menos muchos menos que los procesadores de otras arquitecturas. Esto le permite ofrecer un rendimiento aceptable con un consumo energético muy bajo, y no sólo eso: fabricar un procesador ARM también es notablemente más barato, lo cual los hace ideales en muchos casos.

**Lo que nos espera con ARM**

Desde el nacimiento de Advanced RISC Machines Ltd hace poco más de veinte años, ARM ha crecido de forma muy notable tanto económica como tecnológicamente. Ya hemos visto que es una arquitectura que se utiliza en prácticamente cualquier dispositivo electrónico moderno, e incluso muchos elementos a priori más tradicionales están adoptando esta tecnología para ofrecer nuevas funciones a los usuarios: por ejemplo los nuevos aspiradores robot (el iRobot Roomba 570 utiliza un ARM7TDMI bajo la arquitectura ARMv4) o incluso los juguetes más tecnológicamente avanzados (Lego Mindstorms NXT también integra el ARM7TDMI en sus entrañas).

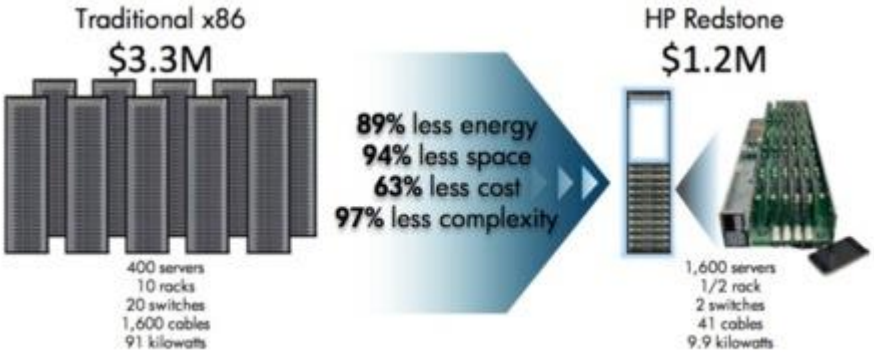



Diagrama del proyecto de servidores ARM de HP, el HP Moonshot

En el actual panorama de crisis económica y con un cuidado por la naturaleza cada vez mayor, ARM es una opción que se está estudiando para su uso en grandes centros de datos como sustituto de las actuales arquitecturas. Su principal finalidad es la de abaratar los costes relacionados con el enorme consumo energético, no sólo por los propios computadores sino también por los ingentes sistemas de ventilación y seguridad asociados. ARM no sólo consume menos energía sino que también genera menos calor, otra característica muy importante en los datacenters. Ya existen varios ejemplos a lo largo de todo el mundo que han creado sistemas de unos cuantos cientos de procesadores, y si bien no son los más potentes (no hay ningún ARM en el Top500 de supercomputadores) sí ofrecen un rendimiento más que digno para muchas, muchísimas de las tareas más profesionales de la actualidad. Es cuestión de tiempo que ARM entre en esa prestigiosa y conocida lista.

A día de hoy podemos asegurar que el futuro de ARM es muy prometedor. Van ganando rendimiento en cada nueva generación de procesadores, van manteniendo - o incluso reduciendo - el consumo energético, y por si fuera poco la innovación tiene muy en cuenta a esta arquitectura. El concepto de casa inteligente se basará en procesadores ARM, los coches están empezando a utilizarlos - y en no mucho tiempo será algo común en nuestros vehículos - además de por supuesto nuevos conceptos de gadgets que aparecerán en los próximos años de la misma forma que hace no mucho aterrizaron los teléfonos avanzados, las tabletas o los libros electrónicos.

Y no solo eso: los usuarios están modificando su concepto de ordenador personal. Estamos ante eso que muchos denominan la era post-PC en la que ARM puede ser vital para conseguir nuevos tipos de dispositivos portátiles y con autonomías hoy impensables, pero que serán razonables y tecnológicamente viables dentro de unos pocos años.

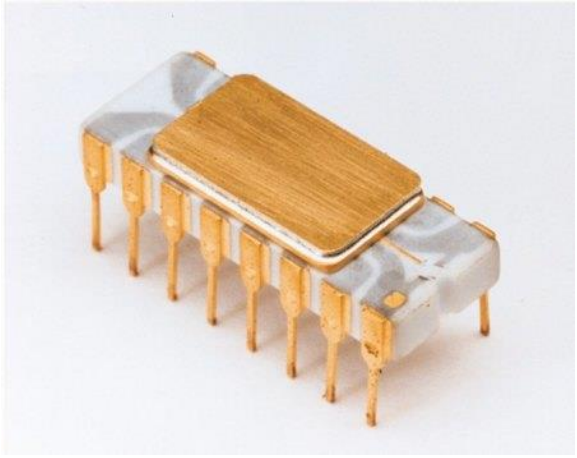
## Arquitectura x86

Conocer la arquitectura x86 es conocer la base de los procesadores de la todavía breve historia de la informática. Se cuentan por millones los procesadores vendidos a lo largo de la historia que la implementan, y además ha sido una arquitectura que dio pié al éxito de compañías como Microsoft e Intel

así como al concepto de ordenador personal. x86 es un hito tecnológico que merece la pena ser analizado y estudiado con gran profundidad.

A continuación la conoceremos a fondo, desmigaremos su historia, la situaremos en el mercado y explicaremos las características técnicas de la más notable arquitectura CISC del mercado. x86 es una de las creaciones más importantes en el mundo del hardware, y conocerla es imprescindible para comprender la historia de la informática y su estado actual. Así que pasen y lean, están todos invitados

## Un poco de historia



**Intel 4004**

En 1971 Intel presentó el primer procesador comercial del mercado, el 4004. Se trataba de un modelo con bus de 4 bits y pensado para su uso en calculadoras, muy vetusto pero considerado como la primera piedra de todo lo que se presentaría en esa década. A él lo siguieron los Intel 8008 y 8080 en 1972 y 1974, respectivamente, con set de instrucciones diseñado por Datapoint Corporation y también pensados para ser utilizados en las calculadoras más avanzadas de la época.

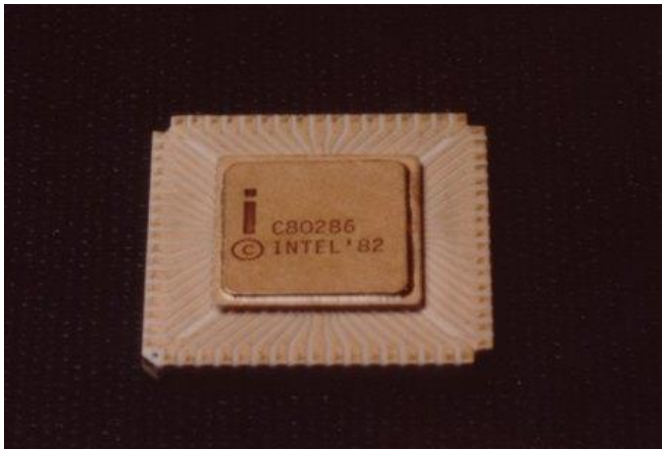
Tras estos modelos Intel empezó un enorme proyecto con el que buscaba reinventar el mundo de los procesadores. En 1975 inició los diseños de la arquitectura iAPX 432 (Intel Advanced Processor Architecture) de 32 bits, con mejoras en la multitarea y la administración de memoria respecto de la familia de los 8000, siendo una arquitectura pensada en la programación orientada a objetos y con la capacidad de administrar múltiples procesos simultáneamente. Sin embargo, viendo que la competencia preparaba sus respectivos nuevos productos (Motorola 6800, MOS 6502 o Zilog Z80) y teniendo en cuenta lo ambicioso del proyecto 432 que les obligaría a estar varios años sin presentar un nuevo producto, decidieron iniciar un nuevo trabajo que, si bien en un inicio podía ser considerado temporal, fue lo que le ha mantenido ocupados durante todas estas décadas.


**Intel 8086, 1979**

En el mes de mayo de 1976 comenzaron a desarrollar un nuevo procesador que empezó a venderse un par de años más tarde, a mediados de 1978. El Intel 8086 incluyó retrocompatibilidad con el software de los anteriores 8008 y 8080, utilizando transistores de 3 micras (unas 135 veces más grandes que los actuales de 22 nanómetros) y una frecuencia de hasta 4.77 MHz. Lo más importante de todo fue la introducción de un nuevo juego de instrucciones diseñado por Intel y denominado x86-16. Fue un pequeño éxito para Intel, y si bien era realmente potente para la época su precio se consideró excesivo. Uno de los puntos más curiosos del 8086 es que si bien fue un invento de Intel, otras compañías como OKI, Siemens, Fujitsu o AMD distribuyeron en el mercado clones de este modelo plenamente compatibles y que incluso en algunos aspectos mejoraban las características del original.

Unos meses más tarde, el 1 de julio de 1979 se presentó el Intel 8088, una versión más barata del 8086 con prácticamente todas sus características, y que tuvo en IBM a su principal aliado. El IBM PC 5150, considerado el primer ordenador personal ('PC') de venta masiva de la historia, utilizó el 8088 en detrimento de otros procesadores de la competencia debido a su alta disponibilidad, facilidad de programación y reducido precio. El IBM PC, que rondaba los 3.000 dólares, fue un éxito de ventas desde su lanzamiento, el 12 de agosto de 1981.

Fue entonces cuando Intel posicionó como su principal producto una nueva familia de chips: los procesadores 'x86', definidos así por usar el mismo juego de instrucciones que, si bien ha ido evolucionando con los años, sigue utilizando muchas de las características originales.




**Intel 80286, 1982**

El 8086 es considerado el primer x86 de la historia, si bien el modelo que catapultó la fama de Intel fue el 8088. A él le sucedieron los 80186, 80286, 80386 y 80486, más conocidos como 186, 286, 386 y 486 que fueron presentados a lo largo de la década de los 80 y principios de los 90 dentro de equipos englobados bajo el concepto que fué creado en el IBM PC, el cual supuso un completo éxito. Después de los 80X86 Intel se movió a los conocidísimos Pentium, con el primer Pentium en 1993 seguido de Pentium Pro, Pentium II, Pentium III y Pentium 4. Incluso los actuales Ivy Bridge mantienen la marca 'Pentium' en alguno de sus modelos de más baja gama.


**Intel Pentium, 1992**

Por su parte, aquella arquitectura iAPX 432 que tantas expectativas generó en su gestación fue finalmente presentada en 1981, pero defraudó enormemente a los usuarios quienes consideraron que ofrecía un rendimiento muy por debajo de lo esperado. Intel la abandonó pocos meses después de presentarla al público.

Desde el primer x86 allá por los años setenta, Intel ha sido la encargada de mover la batuta de la que es considerada la principal arquitectura CISC de la historia, si bien no es la única: Motorola, DEC y sobre todo IBM han presentado procesadores CISC a lo largo de la historia. Sus principales características las describiremos más adelante, pero antes nos pararemos a hablar sobre un tema importante: las licencias x86.

## x86 y las licencias

ARM diseña sus arquitecturas y vende esta propiedad intelectual a otras compañías, encargadas de tomar esos diseños, modificarlos si fuese necesario y luego ordenar la fabricación del procesador final. Intel toma un doble papel de diseñador y fabricante de sus procesadores, ya que también vende las licencias a otros fabricantes.

El ejemplo más claro de compañía a la que Intel le licencia a arquitectura x86 lo encontramos en AMD, quien desde hace varias décadas acuerda con Intel un contrato tal que le permite fabricar chips con el mismo juego de instrucciones, de forma que son plenamente compatibles en software aunque utilizan hardware diferente. En la actualidad sólo AMD y VIA tienen licencias x86 proporcionadas por Intel, además de lógicamente ella misma. Las cuotas de mercado hablan por si mismas:

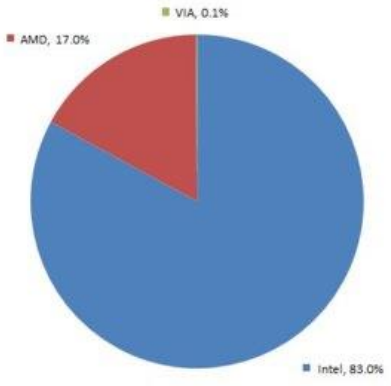



Diagrama de porcentajes de licencias x86

## Intel X86 y sus principales características

La larga experiencia de la compañía ha repercutido en múltiples y notables cambios en muchos aspectos, que van desde la mejora en el tamaño de los buses de datos (16, 32 y ahora 64 bits) a múltiples nuevas instrucciones añadidas que han ido adaptándose a los nuevos usos de la tecnología

Por ejemplo el 8086 no disponía de operaciones en coma flotante, pera lo cual Intel creó un coprocesador matemático que realizase estas operaciones: fue una pequeña extensión sobre x86 denominada x87 y lanzada al mercado bajo los nombres 8087, 80187, 80287, 80387, 80487 y la última 80587, ya a mediados de los 90. Se trataba de procesadores independientes que proporcionaban un extra de rendimiento en cierto software, pero que a la vez suponían un coste adicional respecto del equipo original que tampoco era considerado muy barato. Desde hace un par de décadas todos los procesadores x86 comerciales incluyen instrucciones para operar en coma flotante, con lo que no es necesario un coprocesador adicional salvo en ciertos usos muy concretos en los que se requiere de una gran potencia de cálculo, por ejemplo con sistemas como NVidia Tesla.

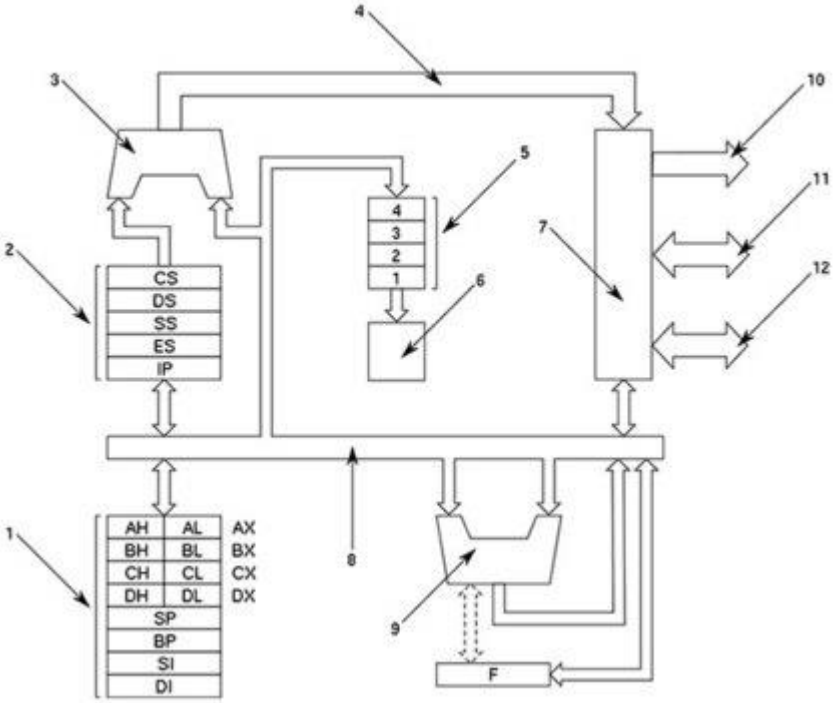



Diagrama esquemático del Intel 8086

x86 también ha ido incluyendo nuevos sets de instrucciones y mayores registros para afrontar todo tipo de tareas. MMX, que añadió los registros utilizados en la arquitectura x87, ha sido uno de los más conocidos y fue incluido en un conjunto de modelos basados en los Pentium y lanzados en 1996, si bien hay muchos más: SSE (Streaming SIMD Extensions en sus versiones SSE, SSE2, SSE3, SSSE3 y SSE4; introdujo múltiples instrucciones para operar con datos en coma flotante, enteros o posiciones de memoria), 3DNow! (una evolución sobre MMX, también para aplicación gráfica) o AES (más reciente, para cifrado de la información). En Wikipedia puede consultarse un breve resúmen de las instrucciones disponibles, si bien la documentación más completa la proporciona la propia Intel y se compone de unos cuantos miles de páginas.

Si echáis un vistazo a ese par de enlaces veréis que en x86 dispondremos de cientos de instrucciones, muchas de ellas complejas que podrían subdividirse en varias instrucciones más pequeñas. Sin embargo la filosofía de CISC es precisamente esa: proporcionar un amplio conjunto de instrucciones que pueden abarcar múltiples tareas más simples. El algoritmo del huevo frito de nuestra primera entrada introductoria era el siguiente:

- Paso 1: Poner la sartén en la vitrocerámica
- Paso 2: Echar aceite

- Paso 3: Calentar el aceite
- Paso 4: Esperar a que esté caliente
- Paso 5: Cascar el huevo
- Paso 6: Verterlo con cuidado sobre el aceite caliente
- Paso 7: Con la ayuda de una paleta, echar el aceite por encima del huevo
- Paso 8: Comprobar que el huevo ya está cocinado y, en ese caso, sacarlo a un plato

Supongamos que cada uno de esos 'pasos' puede ejecutarse en un procesador Intel 8088. Sin embargo, jugando a ser Intel procedemos a lanzar el Pentium MMX con nuevas instrucciones entre las que se encuentran dos nuevas: [Echar y calentar el aceite] y [Cascar el huevo y verterlo con cuidado sobre el aceite caliente], ambas más complejas que las anteriores. Si las utilizásemos en nuestro algoritmo el resultado sería el siguiente:

- Paso 1: Poner la sartén en la vitrocerámica
- Paso 2: Echar y calentar el aceite
- Paso 4: Esperar a que esté caliente
- Paso 5: Cascar el huevo y verterlo con cuidado sobre el aceite caliente
- Paso 7: Con la ayuda de una paleta, echar el aceite por encima del huevo
- Paso 8: Comprobar que el huevo ya está cocinado y, en ese caso, sacarlo a un plato

Unos años más tarde estamos diseñando Pentium 4 y decidimos probar con una nueva instruccion, [Echar y calentar el aceite hasta que alcance su temperatura óptima], con lo que podríamos volver a modificar el algoritmo:

- Paso 1: Poner la sartén en la vitrocerámica
- Paso 2: Echar y calentar el aceite hasta que alcance su temperatura óptima
- Paso 5: Cascar el huevo y verterlo con cuidado sobre el aceite caliente
- Paso 7: Con la ayuda de una paleta, echar el aceite por encima del huevo
- Paso 8: Comprobar que el huevo ya está cocinado y, en ese caso, sacarlo a un plato

Podríamos continuar añadiendo nuevas instrucciones cada vez más complejas para completar nuestra tarea, minimizando el número de pasospero siendo estos cada vez más complejos. Incluso llevando la filosofía al extremo, si vemos que nuestros usuarios/clientes hacen muchos huevos fritos en sus desarrollos podríamos implementar una única instrucción:

- Paso 0: Hacer un huevo frito

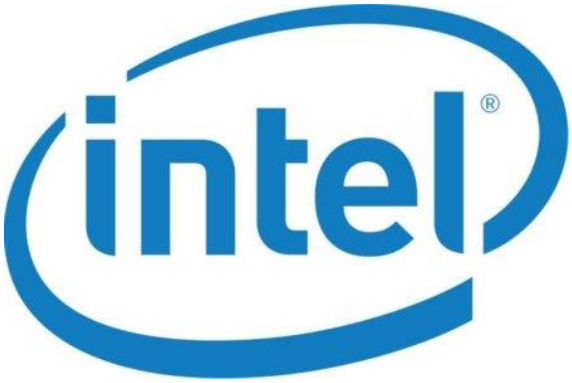
Que sería la más compleja de todas.

Otro aspecto fundamental de los procesadores x86 es su rendimiento. A lo largo de la historia ha demostrado ser una arquitectura muy potente, pero esto tiene su contrapartida: son un conjunto de procesadores que consumen más energía que los modelos ARM.

### Arquitectura x86, conclusiones

Terminamos este capítulo de nuestro especial de arquitecturas con algunas conclusiones finales sobre x86, una de las más importantes de la historia.

El rendimiento de x86 ha crecido sustancialmente con cada nueva generación de procesadores. Intel, como gran creadora de la arquitectura ha ido adaptándola a los nuevos usos por parte de los usuarios, añadiendo no sólo nuevas instrucciones sino también nuevos chips. Lo último ha sido introducir una unidad de procesamiento gráfico junto a la CPU, lo cual permite abaratar costes y ofrecer un conjunto que si bien no es excesivamente potente en el procesamiento de imágenes, sí es suficiente para muchos usuarios.



El futuro de la arquitectura x86 mezclará su negocio tradicional en equipos de sobremesa y portátiles junto a esa nueva categoría de dispositivos 'portátiles', móviles y tabletas principalmente. Intel ya ha introducido su tecnología en algunos dispositivos con unos remodelados Intel Atom de bajo consumo, y a lo largo de este año está confirmado su uso en muchos otros gadgets de diversas marcas.
Por otro lado, el gran rival de x86 es ARM, una arquitectura cuyo crecimiento en los últimos años está siendo importantísimo al conseguir introducirse en dispositivos pequeños, eficientes y con un precio generalmente más atractivo, que seducen a muchos usuarios. No cabe duda de que Intel continuará evolucionando su principal arquitectura hacia los nuevos usos de la tecnología y que seguiremos hablando de ella durante muchos años más.

## Por qué los núcleos y su frecuencia no lo son todo para un procesador

Un procesador es mucho más que frecuencias o núcleos. Tiene transistores, circuitería, electrónica y algoritmos. Incluye hardware y software, y el resultado de toda esta mezcla apenas ocupa unos pocos centímetros cuadrados que muchas veces pasan inadvertidos ante nuestros ojos.
Porque un procesador de un núcleo a 2 Ghz. puede ser más eficiente que otro de cuatro núcleos a 4 GHz. Entran en juego otros componentes (caché, registros) así como los importantísimos algoritmos de ejecución. Por cierto, ¿sabes cómo se haría una tortilla de patata de forma paralela? Lo explicamos tras el salto.

## Secuencial frente a paralelo

La evolución de la tecnología en los últimos años ha llevado al mercado doméstico la importancia de la paralelización de los algoritmos. Tras el ingente crecimiento de la frecuencia de los núcleos a partir de la década de los 90, los fabricantes y diseñadores se han encontrado con límites físicos que les han obligado a acudir a otras vías: la principal de ellas es la implementación de varios núcleos en un mismo chip.
La tortilla de patata española es un gran ejemplo para explicar el concepto de algoritmo paralelo
Un algoritmo secuencial es aquel que sigue un orden concreto. Nuestros ejemplos del algoritmo del huevo fritoson perfectos de esta tarea secuencial al estar organizados en pasos que han de ser seguidos uno tras otro: primero el 1, una vez terminado seguiremos con el 2, tras él el 3, etc.
Modifiquemos la receta y añadámos algo de complejidad. El algoritmo de la tortilla de patata española, un plato clásico por estos lares y conocido en todo el mundo que nos permitirá explicar también el concepto de algoritmo paralelo en contrapartida al secuencial.

En primer lugar es necesario especificar cuál es nuestro algoritmo secuencial de la tortilla de papa:

1 Preparamos cuatro papas medianas

2 Preparamos una cebolla de tamaño medio

3 Pelamos las patatas

4 Pelamos la cebolla

5 Las patatas las troceamos en cachos muy finos y pequeños

6 Repetimos el proceso con la cebolla al gusto de los comensales

7 Juntamos la patata y la cebolla en un bol de grandes dimensiones. Removemos el conjunto

8 Preparamos la sartén adecuada

9 Vertemos una generosa cantidad de aceite

10 Calentamos la sartén a fuego medio y esperamos a que coja la temperatura adecuada

11 Vertemos el contenido del bol a la sartén

12 Remover cada tres minutos hasta que la patata esté blanda

13 Extraer el contenido de la sartén a un bol de grandes dimensiones intentando mover la menor cantidad de aceite posible

14 Preparar cuatro huevos. Cascarlos y verter su contenido en otro bol. Batirlo hasta que quede un líquido ciertamente espeso y uniforme.

15 Unir el contenido de los dos boles y mezclar el conjunto hasta que el resultado sea uniforme

16 Calentar de nuevo la sartén, esta vez al máximo.

17 Cuando la sartén haya cogido temperatura, verter el contenido del bol

18 Cuando los bordes de la tortilla estén ya cuajados, darle la vuelta a la tortilla con ayuda de un plato

19 Volver a insertar la tortilla a medio hacer en la sartén para que termine de hacerse por el otro lado

| | | |
|---|---|---|
| 20 | Cuando esté cuajada por completo, sacar a un plato | |

Éste será nuestro algoritmo de la tortilla de patata de hoy. Y nótese que es secuencial, lo que quiere decir que tendremos que completar cada una de las tareas en el orden indicado, una por una, empezando por la primera, terminando en la última e iniciando un paso únicamente cuando se haya terminado el anterior. El algoritmo arriba descrito sería una aproximación a una tarea secuencial ejecutada por un procesador con un único núcleo.
Cambiemos de filosofía. Ahora disponemos de un procesador de dos núcleos, cuyo símil en la vida real sería algo así como que tenemos dos personas en la cocina:

| | Cocinero 1 | Cocinero 2 |
|---|---|---|
| 1 | Preparamos cuatro patatas medianas | Preparamos una cebolla de tamaño medio |
| 2 | Pelamos las patatas | Pelamos la cebolla |
| 3 | Las patatas las troceamos en cachos muy finos y pequeños | Repetimos el proceso con la cebolla al gusto de los comensales |
| 4 | Juntamos la patata y la cebolla en un bol de grandes dimensiones. Removemos el conjunto | Preparamos la sartén adecuada |
| 5 | | Vertemos una generosa cantidad de aceite |
| 6 | | Calentamos la sartén a fuego medio y esperamos a que coja la temperatura adecuada |
| 7 | Vertemos el contenido del bol a la sartén | |
| 8 | Remover cada tres minutos hasta que la patata esté blanda | Preparar cuatro huevos. Cascarlos y verter su contenido en el bol. Batirlo hasta que quede un líquido ciertamente espeso y uniforme. |
| 9 | Extraer el contenido de la sartén al bol de los huevos intentando traspasar la menor cantidad de aceite posible | Calentar de nuevo la sartén, esta vez al máximo. |
| 10 | Verter el contenido del bol a la sartén | |
| 11 | Cuando los bordes de la tortilla estén ya cuajados, darle la vuelta a la tortilla con ayuda de un plato | |
| 12 | Volver a insertar la tortilla a medio hacer en la sartén para que termine de hacerse por el otro lado | |
| 13 | Cuando esté cuajada por completo, sacar a un plato | |

La diferencia es palpable: hemos utilizado 13 filas en vez de 20, que en un procesador sería el equivalente a utilizar trece ciclos de CPU en vez de veinte: en torno a un 35% menos instrucciones, lo cual es mejora muy significativa. Si cada una de las instrucciones (filas) se ejecutase en el mismo tiempo, nuestro algoritmo sería un 35% más rápido.
Hay tareas que no se pueden paralelizar; otras, al contrario, son fácilmente paralelizables.

Gracias a este sencillo ejemplo comprobamos algunos de los problemas de la ejecución en paralelo: hay tareas que no se pueden paralelizar, por ejemplo las tareas 5 y 6 de nuestro segundo algoritmo deben ser secuenciales, ya que se necesita que primero vertamos el aceite y luego calentemos la sartén (o de lo contrario nos quemaremos). Al contrario, hay otras tareas que son fácilmente paralelizables: mismamente las que nos encontramos en 8, remover y preparar los huevos. Podemos hacer una cosa mientras la otra persona prepara la otra.
Como conclusiones, en primer lugar, dos núcleos no significa que mejoremos por dos el rendimiento del procesador de un núcleo, ni cuatro núcleos que lo mejoremos por cuatro. Hay tareas que no pueden ser paralelas y que necesariamente deben ser secuenciales. Por otro lado es necesario tener en cuenta que los procesadores se encargan de gestionar cientos o miles de tareas simultáneamente, de la misma forma que en una cocina se suelen crear varios platos simultáneamente y, por ello, esos huecos que en la tabla se ven vacíos podrían aprovecharse en otras recetas.

## La ejecución fuera de orden: adelantando trabajo

La ejecución fuera de orden es un concepto que silenciosamente hemos introducido en nuestro ejemplo del algoritmo paralelo de la tortilla de patata. Básicamente consiste en adelantar trabajo que prevemos vamos a realizar posteriormente. Aunque no tengamos todos los elementos necesarios para continuar con nuestra receta, sí podemos hacer algo que luego nos permitirá ahorrarnos un paso.

La ejecución fuera de orden consiste en adelantar trabajo que prevemos vamos a realizar posteriormente Nótese por ejemplo el paso 8 del algoritmo paralelo. Mientras uno está removiendo el contenido de la sartén, otro puede ir preparado cuatro huevos, cascarlos en un bol e ir batiéndolos. En nuestro ejemplo de algoritmo secuencial teníamos dos pasos diferentes, 12 y 13, de forma que en el primero de ellos estábamos removiendo y esperando en intervalos de tres minutos, y en el otro -- que, recuerdo, sólo comenzaba cuando el anterior había terminado -- batiendo los huevos. Es claramente una tarea que puede hacerse simultáneamente, ya que no interfieren unos elementos con otros.
Un ejemplo más práctico y semejante a la realidad, relacionado con operaciones básicas, es el siguiente:

- Le asignamos al registro A el valor 0 (A:=0)
- Le asignamos al registro B el valor 1 (B:=1)
- Le asignamos al registro C el valor 2 (C:=2)
- Incrementamos en 1 el valor del registro C (C:=C+1)
- Le asignamos al registro D el valor 3 (D:=3)

Éste sería el algoritmo para un procesador mononúcleo. En caso de tener dos núcleos hay algunas de esas cinco tareas que pueden hacerse simultáneamente:

Núcleo 1 Núcleo 2

1 A:=0 B:=1

2 C:=2 D:=0

3 C:=C+1=3

¿A qué se debe ese hueco en blanco? A que hemos adelantado la tarea D:=0ya que para ejecutar la anterior C:=C+1 se necesita que C exista y tenga un valor asignado, o de lo contrario no se puede sumar 1 a un hueco vacío. Este ejemplo simplemente juega con cuatro variables y una suma, con lo que es sencillo a más no poder; en la práctica las tareas son mucho más complejas y el algoritmo OOE es algo diferente: existe una cola de instrucciones pendientes de ejecutarse y un pequeño algoritmo que se encarga de gestionarlas: si pueden hacerse, se hacen; si no pueden hacerse en este preciso instante, se postponen para que el procesador no esté se mantenga a la espera y se ejecutarán posteriormente una vez se disponga de todos sus operandos.

La ejecución fuera de orden (out of orden execution, OoOE, OOE) empezó a gestarse en los años setenta y ochenta, aunque no fue hasta la década de los 90 cuando se popularizó en el mercado. El primer procesador que incluyó esta mejora fue el POWER1 de IBM, si bien a día de hoy es una técnica implementada en prácticamente cualquier microprocesador moderno x86, así como en muchos de los actuales ARM.

## Núcleos, frecuencias y mucho más

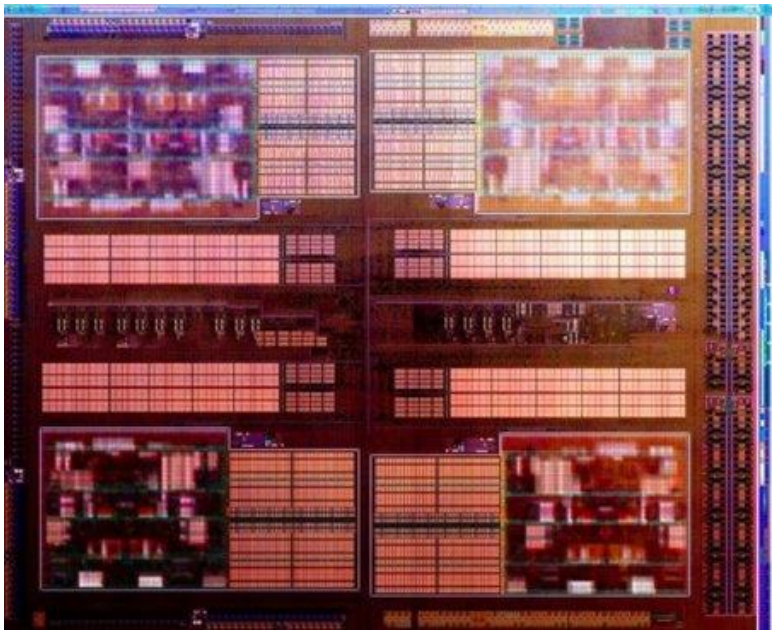
Lo que hemos visto en todos estos especiales es una pincelada de lo que es la realidad. Hemos hablado de arquitecturas, instrucciones, núcleos y algoritmos, hemos mencionado OOE e incluso hemos puesto algunos ejemplos prácticos.


Lo que tenéis aquí arriba es la circuitería de un procesador, en este caso un Intel Core 'Ivy Bridge'. A continuación os dejo una fotografía de AnandTech de las tripas del SoC Apple A6 utilizado en el iPhone 5:

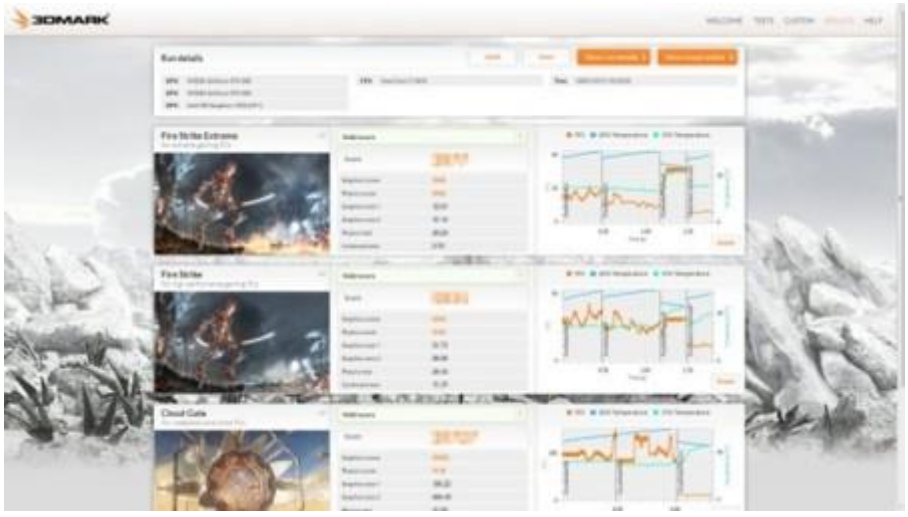

Un procesador tiene mucho más que núcleos y frecuencias. Si ahondamos en el tema nos encontraremos con mucha memoria distribuida por doquier, y no sólo memoria caché (L2, L3; se encargan de comunicar unos núcleos con otros) si no también registros o contadores, así como buses de datos o circuitos de toda índole. El conjunto de todos estos componentes muchas veces microscópicos formarán un chip de pequeñas dimensiones (unos pocos centímetros cuadrados) que es al que denominamos procesador.

**¿De qué depende el rendimiento?**

Es sin duda alguna una pregunta que debemos hacernos. ¿De qué depende realmente el rendimiento de un procesador? ¿Por qué un procesador a 1.6 GHz. puede ser más potente que uno a 4 GHz.?
Como hemos visto, un microprocesador es mucho más que una frecuencia. Ésta indica la periodicidad con la que la CPU ejecuta una instrucción, cuya definición ya vimos en una de las primeras entradas. A priori podríamos pensar que a mayor frecuencia de funcionamiento es mejor, pero también entran en juego otros muchos factores -- tanto hardware (caché, su localización y cantidad) como software (algoritmos como el OOE) -- que no sólo repercuten en el rendimiento final, si no también en el precio de mercado del chip. Este último factor es fundamental por ejemplo en entornos profesionales, donde se estudia cada céntimo de inversión.



¿Veis los cuatro núcleos acompañados de las ocho (4xL2 y 4xL3) caché? Éste es un AMD FX
Con el número de núcleos ocurre algo parecido. Cuantos más núcleos tengamos deberíamos obtener un mayor rendimiento, pero también es necesario que vengan acompañados de la circuitería necesaria para que trabajen de una forma eficiente. No valdrá de nada tener una CPU de diez núcleos si no existe un algoritmo y un buena base electrónica que se encarguen de situar las instrucciones en cada una de las colas de proceso de cada núcleo, así como por supuesto gestionarlas correctamente. Ésta era la razón por la que los primeros procesadores multinúcleo de uso doméstico, lanzados hace unos cinco años, no funcionaban tan bien como deberían; en la actualidad se ha avanzado notablemente y el paralelismo en tareas domésticas es bastante aceptable.
Para nosotros, los usuarios, siempre es interesante estudiar el rendimiento de un procesador como un todo, como un conjunto de componentes cuyo resultado es la ejecución de un programa en nuestro ordenador. Para ello sería ideal poder determinar a priori y sólo con las características técnicas del chip una estimación del rendimiento. Lamentablemente o existe ninguna técnica que cumpla estos requisitos y la mejor forma de atacar este problema es acercándonos en la medida de lo posible a su realidad: utilizar software específico. En este ámbito recibe un nombre conocido por todos: benchmarks. Hace no mucho hablábamos de benchmarks para móviles, cuyo funcionamiento es similar al de sus homólogos para plataformas de escritorio, como por ejemplo Windows.



Captura de los resultados de 3DMark, uno de los benchmarks más utilizados

Un benchmark es como cualquier otro software. Se programa y ejecuta una serie de tareas. A diferencia de los programas a los que estamos acostumbrados, un benchmark ejecuta siempre la misma secuencia de código, de forma que sus resultados sobre un mismo equipo deberían ser siempre los mismos. Las mínimas variaciones que se producen son debidas a que el estado del sistema operativo en cada momento es sensiblemente diferente, por ejemplo debido a la ejecución de pequeños procesos en segundo plano.
Un benchmark ejecuta siempre el mismo código independientemente del hardware, con lo que los resultados pueden compararse entre diferentes máquinas.
La clave de los benchmarks y su principal característica es precisamente la de ejecutar el mismo código, independientemente del hardware. Al ejecutar exactamente lo mismo se utilizan los mismos recursos de los componentes hardware, con lo que los resultados pueden compararse entre una máquina y otra.
No obstante, un benchmark no es un tema baladí. Existen decenas de tipos de benchmarks que están centrados en los diferentes componentes: gráficos, de potencia bruta, para el almacenamiento o de red. Si bien existen benchmarks genéricos, estos generalmente se componen de varios benchmarks específicos que sacan el máximo partido a cada componente: GPU, CPU, memoria RAM, etc. A cada uno de ellos se le da un peso concreto y posteriormente se realiza una media, dando lugar a una cifra final que nos indica su "nota" y que puede fácilmente compararse con otros resultados.
No puedo terminar este apartado sin mencionar que estamos muy acostumbrados a las características técnicas, pero en el mundo actual con los componentes tan complejos que tenemos éstas no nos lo dicen todo. La única opción real que tenemos para medir la diferencia de rendimiento entre dos procesadores son los benchmarks. Los hay mejores y peores, más o menos completos; pero es la única herramienta fiable que tenemos a nuestra disposición.

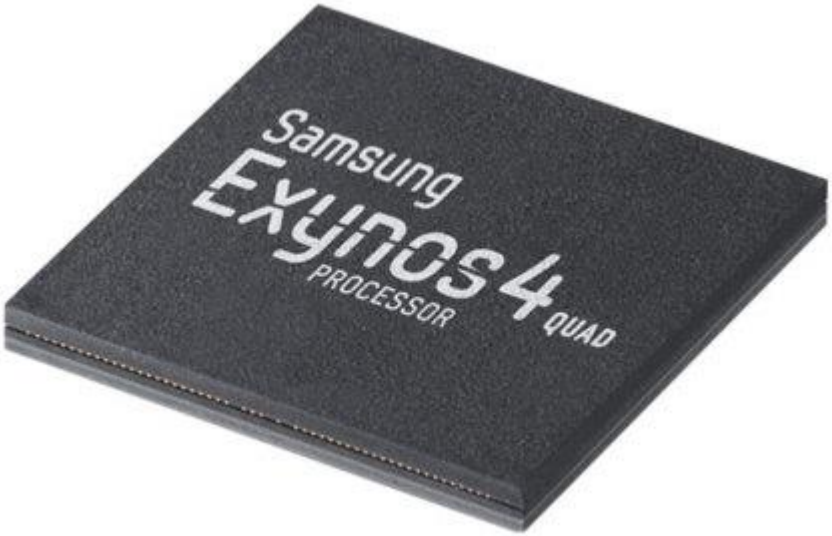
## Un ejemplo real

Enfilamos ya la recta final con un ejemplo real y práctico de que los núcleos y su frecuencia no lo son todo. En la siguiente tabla encontraréis las características de dos teléfonos de primera línea presentados en los últimos meses, ambos basados en ARM:

| | Teléfono 1 | Teléfono 2 |
|---|---|---|
| Set de instrucciones | ARMv7 | ARMv7 |
| Núm. núcleos | 2 | 4 |
| Tipo de núcleos | Krait | Cortex-A9 |
| Frec. núcleos | 1.7 | 1.6 |
| RAM | 1 GB | 2 GB |
| Resolución de la pantalla | 1280x720 | 1280x720 |
| Sistema operativo | Android 4.1 | Android 4.1 |

Con el fondo rojo están las características inferiores a sus homólogas verdes del otro teléfono, lo que a priori podríamos pensar es mejor o peor. Con estos datos es lógico pensar que la balanza se inclina a favor del Teléfono 2: la menor frecuencia de sus núcleos (sólo un 0.1 GHz. menos) no se impone debido a que tiene el doble de ellos y que además duplica la cantidad de memoria RAM. Ambos son ARMv7 aunque varía el fabricante y el diseño del núcleo, un Krait y un Cortex-A9. A pesar de ello ambos implementan la ejecución fuera de orden de la que hablábamos anteriormente.

Así pues, el Teléfono 1 tiene todas las de perder. Hagamos un ejercicio mental: ¿cuál sería la mejora estimada para el Teléfono 2? ¿Con qué porcentaje le ganaría a nuestro primer participante? Pensemos durante unos segundos y, cuando estéis listos para conocer la solución, pasad las siguientes imágenes.


Efectivamente el Teléfono 1 utiliza un Qualcomm Snapdragon, concretamente un S4 Pro MSM8960T; por su parte, el Teléfono 2 trae consigo un Samsung Exynos 4 Quad 4412. Ambos procesadores fueron presentados a lo largo de 2012, con lo que son tecnologías modernas. En el caso que aquí nos concierne se trata de los teléfonos Sony Xperia SP y Samsung Galaxy Note II.

Y finalmente, ¿cuál de los dos es más potente? ¿Piensas que el Samsung Galaxy Note II ofrece un mayor rendimiento que el Sony Xperia SP?. Tal vez hayas cambiado de parecer tras conocer los nombres de nuestros protagonistas... o tal vez no. No te entretengo más, en la siguiente tabla están un trio de benchmarks que hemos ejecutado en los dos terminales:

| | Sony Xperia SP | Samsung Galaxy Note II | % de mejora |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Lanzamiento | 04/2013 | 09/2012 | - |
| CPU | Qualcomm MSM8960T Snapdragon S4 Pro | Samsung Exynos 4412 Quad | - |
| Núm. núcleos | 2xKrait ARMv7 | 4xCortex-A9 ARMv7 | - |
| Frec. núcleos | 1.7 | 1.6 | - |
| GPU | Adreno 320 | Mali-400MP | - |
| RAM | 1 GB | 2 GB | - |
| AnTuTu benchmark | 15117 | 13473 | +12,20% |
| Quadrant Standard | 7649 | 5472 | +39.78 |
| 3DMark | 10236 | 3346 | +205.91% |

Efectivamente, Sony Xperia SP, que sobre el papel parecía el menos potente, sale como ganador de nuestra comparativa y de una forma abrumadora con una media del 85% de mejora respecto del Galaxy Note IIen estos tres tests. Seguro que ahora se entiende perfectamente por qué los núcleos y su frecuencia no lo son todo para un procesador.

# 11. Concurrencia de procesos

La concurrencia de procesos se refiere a las situaciones en las que dos o más procesos puedan coincidir en el acceso a un recurso compartido o, dicho de otra forma, que requieran coordinarse en su ejecución. Para evitar dicha coincidencia, el sistema operativo ofrece mecanismos de arbitraje que permiten coordinar la ejecución de los procesos.

# El problema del productor y consumidor

Un ejemplo de un problema de concurrencia sería el siguiente: Dados dos procesos A (productor) y B (consumidor), suponiendo que ambos se ejecutan indefinidamente en el tiempo, el proceso A debe recibir tiempo de ejecución antes que B, tras esto, el proceso B debe recibir su oportunidad de ejecución, dando paso de nuevo al proceso A y así sucesivamente, siguiendo un esquema de alternancia estricta.

```
proceso A |-- -- --
          |
proceso B | -- -- --
          |_____________
```

Recuerde que el planificador de procesos, al desconocer la naturaleza de los procesos y sus objetivos, no dispone de información suficiente para garantizar la secuencia de ejecución descrita en el ejemplo anterior. Por tanto, suponiendo que ambos procesos se encuentran en estado preparado, podría seleccionar al proceso B para pasar a estado activo antes de seleccionar al proceso A, situación que no es deseada.

Recuerde que el planificador de procesos, emplea criterios de planificación que no tienen en consideración el objetivo de los procesos. Podemos decir que el planificador de procesos **desconoce** cuál es el propósito de los procesos, únicamente puede observar su comportamiento, es decir, si presentan un comportamiento más o menos interactivo.

Por tanto, el programador, a la hora de modelar los procesos, debe emplear mecanismos de arbitraje que ofrece el sistema operativo y que permiten resolver el problema de concurrencia que se plantee.

## Mecanismos de arbitraje

Los mecanismos de arbitraje que ofrece el sistema operativo son básicamente dos:

- Mecanismos de **sincronización**: el sistema operativo ofrece mecanismos que permiten a los procesos coordinar su ejecución para conseguir el objetivo sin que sucedan situaciones no deseadas, como por ejemplo que dos o más procesos coincidan simultáneamente en el acceso a un cierto recurso que no se puede compartir.
- Mecanismos de **mensajería**: el sistema operativo ofrece mecanismos de comunicación entre procesos mediante mensajes. El intercambio de mensajes entre procesos permite coordinarlos.

## Programación concurrente

El término programación concurrente se emplea con frecuencia para referirse a un conjunto de programas que funcionan en cooperación.

Hay tres formas de interacción entre procesos cooperativos:

- Concurrencia: Hay un recurso común, si varios procesos modificaran la misma información a la vez , cada uno podría destruir parte del trabajo de los demás. Si lo hacen uno tras otro, en serie, se obtendrá el resultado correcto.
- Sincronización: El Sistema Operativo se encarga de enviarle señales a los procesos para coordinar su evolución y conseguir que progrese armónicamente.
- Comunicación: El S.O. transmite mensajes entre los procesos, que se usan para intercambiar, enviar o recibir información.

Se le llama programación concurrente ya que la concurrencia fue la primera forma de iterar recursos entre procesos cooperativos.

## Sección crítica

Una *sección crítica* se trata de una sección del código que puede ser ejecutada por un único proceso o hilo simultáneamente. Un ejemplo de *sección crítica* es la sección de código en la que se accede a un recurso compartido. Para evitar el acceso simultáneo a la sección crítica se emplean mecanismos que garantizan la *exclusión mutua*.

La exclusión mutua se debe realizar de forma coordinada y eficiente, para ello se requiere:

- No más de un proceso por sección crítica y recurso compartido.
- El mismo proceso no puede usar el mismo recurso compartido indefinidamente.
- Todo proceso debe entrar antes o después a usar el recurso.
- Si el recurso está sin uso, cualquiera que lo requiera dispondrá de él inmediatamente.
- Si hay varios esperando usar el recurso y éste se libera, uno de los que estaba esperando lo usará durante un tiempo determinado.

Para llevar ésto a cabo se necesita un protocolo que indique cuando el recurso está libre y cuando está siendo ocupado.

## Tipos de mecanismos de sincronización

Los mecanismos de sincronización los podemos catalogar en dos categorías:

- Optimistas: Este mecanismo considera que la frecuencia de acceso a un cierto recurso compartido es *baja*. Este tipo tiene mas consumo de memoria, ya que tiene que copiar el recurso compartido y en caso de interferencia en el hilo tiene que volver a ejecutarlo y consume más recursos.
- Pesimistas: Este mecanismo permite coordinar la ejecución de dos o más procesos que acceden al recurso compartido con una frecuencia *alta*.

## Administración de procesos y del procesador.

## Comunicación entre Procesos.

La comunicación entre procesos, en inglés IPC (Interprocess Communication) es una función básica de los Sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de IPC. La IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí. Normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red subyacente.La comunicación se establece siguiendo una serie de reglas (protocolos de comunicación). Los protocolos desarrollados para internet son los mayormente usados: IP (capa de red), protocolo de control de transmisión (capa de transporte) y protocolo de transferencia de archivos , protocolo de transferencia de hipertexto (capa de aplicación).Los procesos pueden estar ejecutándose en una o más computadoras conectadas a una red. Las técnicas de IPC están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC).
El método de IPC usado puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envió de la misma) de la comunicación entre procesos, y del tipo de datos que están siendo comunicados.La comunicación puede ser:

- Síncrona o asíncrona.
- Persistente (persisntent) o momentánea (transient).
- Directa o Indirecta.
- Simétrica o Asimética.
- Con uso de buffers explícito o automático.
- Envío por copia el mensaje o por referencia.
- Mensajes de tamaño fijo o variable.

**Síncrona:** Quien envía permanece bloqueado esperando a que llegue una respuesta del receptor antes de realizar cualquier otra tarea.
**Asíncrona:** Quien envía continúa con su ejecución inmediatamente después de enviar el mensaje al receptor.
**Persistente:** El receptor no tiene que estar operativo al mismo tiempo que se realiza la comunicación, el mensaje se almacena tanto tiempo como sea necesario para poder ser entregado (por ejemplo, un e-mail).
**Momentánea (transient):** El mensaje se descarta si el receptor no está operativo al tiempo que se realiza la comunicación. Por lo tanto no será entregado.
**Directa:** Las primitivas enviar y recibir explicitan el nombre del proceso con el que se comunican.
Ejemplo: enviar(mensaje, A) envía un mensaje al proceso A. Es decir se debe especificar cuál va a ser el proceso fuente y cuál va a ser el proceso destino.
Las operaciones básicas send y receive se definen de la siguiente manera:
send(P, mensaje); envía un mensaje al proceso P (P es el proceso destino).
receive (Q, mensaje); espera la recepción de un mensaje por parte del proceso Q (Q es el proceso fuente).
Nota: receive puede esperar de un proceso cualquiera un mensaje, pero el send sí debe especificar a quién va dirigido y cuál es el mensaje.
**Indirecta:** La comunicación indirecta se implementa mediante puertos, en alguna bibliografía se lo denomina buzones. Es aquella donde la comunicación está basada en una herramienta o instrumento ya que el emisor y el perceptor están a distancia.

**Simétrica:** Todos los procesos pueden enviar o recibir. También llamada bidireccional para el caso de dos procesos.
**Asimétrica:** Un proceso puede enviar, los demás procesos solo reciben. También llamada unidireccional. Suele usarse para hospedar servidores en Internet.
Uso de búfer automático: El transmisor se bloquea hasta que el receptor recibe el mensaje (capacidad cero).

Se conoce por Comunicación entre procesos ó InterProcess Communication (IPC) al conjunto de mecanismos claves en la compartición de datos (intercomunicación) y sincronización entre distintos procesos. Los procesos que se ejecutan concurrentemente pueden ser independientes o cooperativos. Los procesos son independientes cuando no pueden alterar o verse alterados por otros procesos que están en ejecución. En cambio, los procesos cooperativos pueden verse afectados y afectar a los demás procesos. Razones por las cuales proporcionar entornos que permitan la cooperación entre procesos:

- Compartir información: Ante una eventual necesidad que múltiples usuarios necesiten la misma información (Archivo compartido), se debe proveer un entorno que habilite el acceso concurrente a dicha información
- Acelerar los cálculos: Al desear que una tarea se ejecute mas rápido, se divide en subtareas, implementando un paralelismo de ejecución entre si. Esto se consigue con con computadoras de múltiples elementos de procesamiento, ya sea a nivel de cpu's o canales de E/S.
- Modularidad: Al querer construirse un sistema de forma modular, se necesita dividir la información del sistema en distintos procesos, o hebras.
- Conveniencia: Ya sea para un solo usuario, puede éste querer trabajar en varias tareas simultáneamente. Por ejemplo usando editor de texto, escuchando música e imprimiendo al mismo tiempo.

Dentro de los mecanismos más usados, tenemos el paso por mensajes, la memoria compartida y los semáforos. El paso por mensajes se puede ver como una lista enlazada de mensajes dentro del espacio de direccionamiento del núcleo. Una aplicación, siempre que tenga los derechos necesarios, puede depositar un mensaje (de cualquier tipo) en ella, y otras aplicaciones podrán leerlo. Es posible asignar atributos a los mensajes, de forma que se puedan mantener ordenados por prioridad en lugar de por orden de llegada. La memoria compartida es un medio que permite establecer una zona común de memoria entre varias aplicaciones. Y los semáforos, que son una herramienta puramente de sincronización. Permiten controlar el acceso de varios procesos a recursos comunes.

## Memoria Compartida

El mecanismo de memoria compartida permite a dos o más procesos compartir un segmento de memoria, y por consiguiente, los datos que hay en él. Es por ello el método más rápido de comunicación entre procesos. Al ser el objetivo de este tipo de comunicación la transferencia de datos entre varios procesos, los programas que utilizan memoria compartida deben normalmente establecer algún tipo de protocolo para el bloqueo. Este protocolo puede ser la utilización de semáforos, que es a su vez otro tipo de comunicación (sincronización) entre procesos. La memoria que maneja un proceso, y también la compartida, va a ser virtual, por lo que su dirección física puede variar con el tiempo. Esto no va a plantear ningún problema, ya que los procesos no generan direcciones físicas, sino virtuales, y es el núcleo (con su gestor de memoria) el encargado de traducir unas a otras.

## Utilización

Para aprovechar las ventajas que aporta el uso de memoria compartida se dispone de 4 llamadas al sistema: shmget(), shmctl(), shmat() y shmdt(). En este primer apartado se explicará de una forma general la utilidad de estas llamadas, para ser luego vistas con todo detalle en el siguiente apartado.

### shmget()

Permite acceder a una zona de memoria compartida y, opcionalmente, crearla en caso de no existir. A esta llamada se le pasan tres argumentos: una clave, el tamaño del segmento a crear y el flag inicial de operación, y devuelve un entero, denominado shmid, que se utiliza para hacer referencia a dicho segmento.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

Notar que shmget() únicamente reserva el espacio necesario para alojar el segmento. El segmento no se crea hasta que algún proceso se asigne a él. La posición de memoria final la decide, en principio, el sistema. Si se utiliza para obtener una referencia a una zona de memoria compartida ya existente, el tamaño especificado debe ser inferior o igual al de la memoria existente. En caso contrario, el tamaño asignado debe ser múltiplo de PAGE_SIZE, que corresponde al tamaño de una página de memoria (4 KB en la arquitectura x86). El valor de shmflg se compone básicamente con:

- IPC_CREAT: Para crear un nuevo segmento. Si este flag no se precisa, shmget() únicamente buscará el segmento asociado con la clave y comprobará si el usuario tiene permisos para acceder a él.

- IPC_EXCL: Se utiliza junto a IPC_CREAT para asegurar el fallo si el segmento existe.
- mode_flags: Los 9 bits menos significativos del número están reservados para el establecimiento de permisos de acceso. Actualmente los permisos de ejecución no son utilizados por el sistema.

### shmctl()

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl (int shmid, int cmd, struct shmid_ds *buf);
```

Proporciona una variedad de operaciones para el control de la memoria compartida a través de cmd. Algunas de las operaciones disponibles son:

- IPC_STAT: Lee el estado de la estructura de control de la memoria asociada a shmid y lo devuelve a través de la estructura de datos apuntada por buf.
- IPC_SET: Cambia el valor de los siguientes miembros de la estructura de datos asociada a shmid con el correspondiente valor encontrado en la estructura apuntada por buf: shm_perm.uid shm_perm.gid shm_perm.mode
- IPC_RMID: Marca una región de memoria compartida como destruida, aunque el borrado sólo se hará efectivo cuando el último proceso que la adjuntaba deje de hacerlo. Si en el momento en que se realiza la invocación, el segmento aún está asignado a otro proceso, la clave será puesta a IPC_PRIVATE.
- SHM_LOCK: Bloquea la zona de memoria compartida especificada por shmid. Esto quiere decir que el segmento es grabado en memoria, no siendo posible borrarlo a partir de esta operación. Tampoco se podrá hacer swapping sobre él. Esta cmd solo puede ser ejecutada por un proceso que tenga un ID de usuario efectivo igual al del superusuario. De esta forma se garantiza la permanencia de los datos en la memoria.
- SHM_UNLOCK: Desbloquea la región de memoria compartida especificada por shmid. Esta operación, al igual que la anterior, sólo la podrán ejecutar aquellos procesos con privilegios de acceso apropiados.

### shmat()

Es la llamada que debe invocar un proceso para adjuntar una zona de memoria compartida dentro de su espacio de direcciones. Recibe tres parámetros: el identificador del segmento (shmid), una dirección de memoria (shmaddr) y las banderas descritas más adelante. La dirección puede ser NULL; en tal caso el sistema operativo tomará la responsabilidad de buscar una posición de memoria libre. En option se puede especificar:

- SHM_RND: En cuyo caso, el sistema intentará vincular la zona de memoria a una dirección múltiplo de SHMLBA (shmparam.h) lo más próxima posible a la especificada.
- SHM_RDONLY: Hará que el proceso sólo pueda acceder al segmento de memoria en lectura. Si no se precisa, el segmento se vinculará en modo lectura/escritura.

```
#include <sys/types.h>
#include <sys/shm.h>
void* shmat (int shmid, const void *shmaddr, int option);
```

Como resultado de la invocación, devuelve la dirección de comienzo del segmento de memoria asignado y se actualizan los siguientes campos de la estructura shmid_kernel: shm_atim, que recibe la fecha actual; shm_lprid, que recibe el pid del proceso que llama, y shm_nattch, que se incrementa en una unidad.

### shmdt()

Es la operación que realizará el proceso anterior para desvincularse de una zona de memoria compartida. Para ello, deberá precisarse como primer y único parámetro la dirección shmaddr de dicha zona. Esta llamada al sistema actualiza los campos siguientes de las estructura shmid_kernel: shm_dtim, que recibe la fecha actual; shm_lprid, que recibe el pid del proceso que llama y shm_nattch, que queda decrementado en una unidad.

## Semáforos

Un semáforo es un mecanismo de sincronización que se utiliza generalmente en sistemas con memoria compartida, bien sea un monoprocesador o un multiprocesador. Su uso en un multicomputador depende del sistema operativo en particular. Un semáforo es un objeto con un valor entero al que se le puede asignar un valor inicial no negativo y al que sólo se puede acceder utilizando dos operaciones atómicas: wait y signal (también llamadas down o up, respectivamente). Las definiciones de estas dos operaciones son las siguientes:

```
wait(s){
  s = s - 1;
  if (s < 0)
    Bloquear al proceso;
}
signal(s){
  s = s + 1;
  if (s <= 0)
    Desbloquear a un proceso bloqueado en la operación wait;
}
```

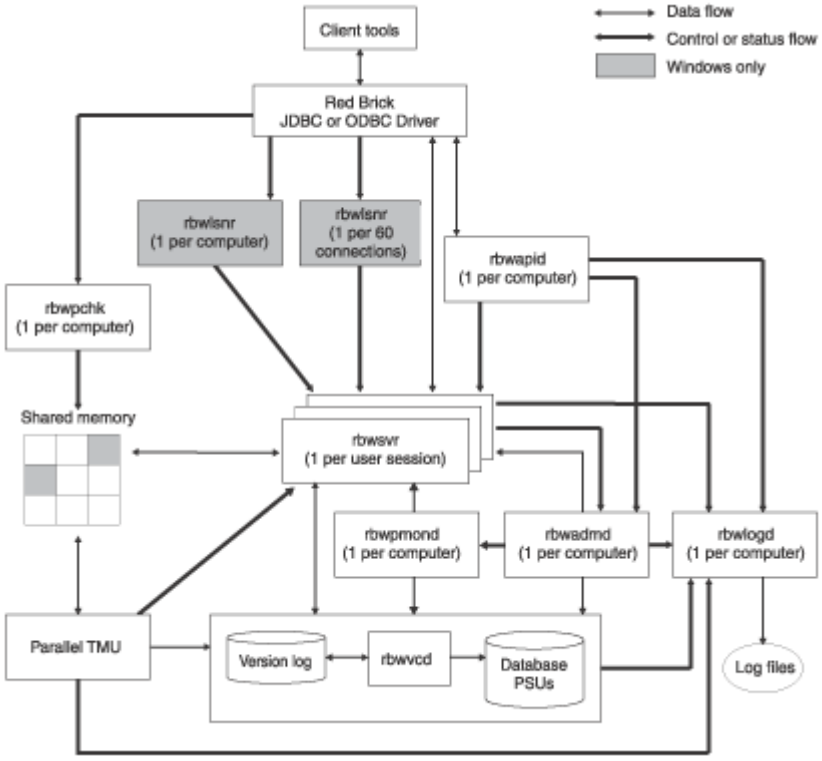
El número de procesos que en un instante determinado se encuentran bloqueados en una operación wait viene dado por el valor absoluto del semáforo si es negativo. Cuando un proceso ejecuta la operación signal, el valor del semáforo se incrementa. En el caso de que haya algún proceso bloqueado en una operación wait anterior, se desbloqueará a un solo proceso. Las operaciones wait y signal son dos operaciones genéricas que deben particularizarse en cada sistema operativo. A continuación se presentan los servicios que ofrece el estándar POSIX para trabajar con semáforos. En POSIX, un semáforo se identifica mediante una variable del tipo sem_t. El estándar POSIX define dos tipos de semáforos:

- **Semáforos sin nombre:** Permiten sincronizar a los procesos ligeros que ejecutan dentro de un mismo proceso o a los procesos que lo heredan a través de la llamada fork.
- **Semáforos con nombre:** En este caso, el semáforo lleva asociado un nombre que sigue la convención de nombrado que se emplea para archivos. Con este tipo de semáforos se pueden sincronizar procesos sin necesidad de que tengan que heredar el semáforo utilizando la llamada fork.

Estos modelos son bastante comunes, y han de usarse en los distintos sistemas operativos, muchos sistemas implementan ambos, el paso de mensajes es útil de implementar en el paso de pequeñas cantidades de información, resulta más fácil de implementar como mecanismo de comunicación entre computadoras. La implementación de memoria compartida resulta tener una velocidad máxima, ya que funciona a nivel de velocidad de memoria, siendo mas rápida que la implementación de paso por mensaje que cuando se implementa se hacen llamadas al sistema, donde se requiere la intervención del kernel el cual consume mas tiempo. En las implementaciones de memoria compartida las llamadas al sistema no se necesitan para establecer las zonas de memoria compartida. Una vez que se establecen las zonas de memorias compartidas, éstas se tratan como zonas de memoria rutinarias para los procesos intervenidos.

En un sistema, los procesos pueden ejecutarse independientemente o cooperando entre sí. Los intérpretes de comandos son ejemplos típicos de procesos que no precisan la cooperación de otros para realizar sus funciones. En cambio, los procesos que sí cooperan necesitan comunicarse entre sí para poder completar sus tareas.

La comunicación entre procesos puede estar motivada por la competencia o el uso de recursos compartidos o porque varios procesos deban ejecutarse sincronizadamente para completar una tarea.




Para que puedan realizarse ambos tipos de interacciones, es necesario que el sistema operativo provea de servicios para posibilitar la comunicación entre procesos.
El sistema operativo provee mínimamente dos primitivas, "enviar" y "recibir", normalmente llamadas send y receive. Asimismo, debe implementarse un enlace de comunicación entre los procesos de la comunicación (pipe). Este enlace puede ser unidireccional o multidireccional según permita la comunicación en uno o en varios sentidos.

# Procesos ligeros (Hilos o hebras).

En la mayoría de los sistemas operativos, estas dos características son, de hecho, la esencia de un proceso. Sin embargo, algunos argumentos pueden convencer de que estas dos características son independientes y que deben ser tratadas de manera independiente por el sistema operativo. Esto se hace así en una serie de sistemas operativos, en particular en algunos sistemas operativos de desarrollo reciente. Para distinguir estas dos características, la unidad de expedición se conoce corno **hilo** (thread) o proceso ligero (lightweight process), mientras que a la unidad de propiedad de los recursos se le suele llamar **proceso** o **tarea.**

Los beneficios clave de los hilos se derivan de las implicaciones del rendimiento: Se tarda mucho menos tiempo en crear un nuevo hilo en un proceso existente que en crear una nueva tarea, menos tiempo para terminar un hilo y menos tiempo para cambiar entre dos hilos de un mismo proceso. Por tanto, si hay una aplicación o una función que pueda implementarse como un conjunto de unidades de ejecución relacionadas, es más eficiente hacerlo con una colección de hilos que con una colección de tareas separadas.

Un hilo de ejecución, en Sistemas Operativos, es similar a un proceso en que ambos representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias. Los hilos permiten dividir un programa en dos o más tareas que corren simultáneamente, por medio de la multiprogramación. En realidad, este método permite incrementar el rendimiento de un procesador de manera considerable. En todos los sistemas de hoy en día los hilos son utilizados para simplificar la estructura de un programa que lleva a cabo diferentes funciones.

Todos los hilos de un proceso comparten los recursos del proceso. Residen en el mismo espacio de direcciones y tienen acceso a los mismos datos. Cuando un hilo modifica un dato en la memoria, los otros hilos utilizan el resultado cuando acceden al dato. Cada hilo tiene su propio estado, su propio contador, su propia pila y su propia copia de los registros de la CPU. Los valores comunes se guardan en el bloque de control de proceso (BCP), y los valores propios en el bloque de control de hilo (TCB).
Muchos lenguajes de programación (como Java), y otros entornos de desarrollo soportan los llamados hilos o hebras (en inglés, threads).
Un ejemplo de la utilización de hilos es tener un hilo atento a la interfaz gráfica (iconos, botones, ventanas), mientras otro hilo hace una larga operación internamente. De esta manera el programa responde más ágilmente a la interacción con el usuario.

Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea.
Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar estos recursos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.
Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros).
El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado. Asimismo en el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

# 12. Interrupciones

En el contexto de la informática, una interrupción (del inglés *Interrupt Request*, también conocida como petición de interrupción) es una señal recibida por el procesador de un ordenador, indicando que debe "interrumpir" el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa, sino que pertenece al sistema operativo o al BIOS. Una vez finalizada dicha subrutina, se reanuda la ejecución del programa.

Las interrupciones surgen de la necesidad que tienen los dispositivos periféricos de enviar información al procesador principal de un sistema informático.

La primera técnica que se empleó para esto fue el *polling*, que consistía en que el propio procesador se encargara de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tenía pendiente alguna comunicación para él. Este método presentaba el inconveniente de ser muy ineficiente, ya que el procesador consumía constantemente tiempo y recursos en realizar estas instrucciones de sondeo.

El mecanismo de interrupciones fue la solución que permitió al procesador desentenderse de esta problemática, y delegar en el dispositivo periférico la responsabilidad de comunicarse con él cuando lo necesitara. El procesador, en este caso, no sondea a ningún dispositivo, sino que queda a la espera de que estos le avisen (le "interrumpan") cuando tengan algo que comunicarle (ya sea un evento, una transferencia de información, una condición de error, etc.).

## Procesamiento

- Terminar la ejecución de la instrucción máquina en curso.
- Salvar el valor del contador de programa, IP, en la pila, de manera que en la CPU, al terminar el proceso, pueda seguir ejecutando el programa a partir de la última instrucción.
- La CPU salta a la dirección donde está almacenada la rutina de servicio de interrupción (Interrupt Service Routine, o abreviado ISR) y ejecuta esa rutina que tiene como objetivo atender al dispositivo que generó la interrupción.
- Una vez que la rutina de la interrupción termina, el procesador restaura el estado que había guardado en la pila en el paso 2 y retorna al programa que se estaba usando anteriormente.

## Clases

## Interrupciones de hardware.

Estas son asíncronas a la ejecución del procesador, es decir, se pueden producir en cualquier momento independientemente de lo que esté haciendo el CPU en ese momento. Las causas que las producen son externas al procesador y a menudo suelen estar ligadas con los distintos dispositivos de E/S.

Las interrupciones de hardware son aquellas interrupciones que se producen como resultado de, por lo general, una operación de E/S. No son producidas por ninguna instrucción de un programa sino por las señales que emiten los dispositivos periféricos para indicarle al procesador que necesitan ser atendidos.

Cuando el microprocesador accede a un periférico (disco duro, puerto de comunicación…), puede transcurrir algún tiempo antes de que los datos sean obtenidos o transmitidos. La solución más simple es esperar hasta recibir los datos o hasta que se haya efectuado la transmisión (polling), pero esta solución bloquea todos los programas en ejecución, y eso no puede admitirse en un sistema multitarea. Por ello, en los sistemas modernos se prefiere un funcionamiento mediante interrupciones, ya que éstas permiten mejorar la productividad del procesador, de forma que este último puede ordenar una operación de E/S y, en lugar de tener que realizar una espera activa, se puede dedicar a atender a otro proceso o aplicación hasta que el dispositivo esté de nuevo disponible, siendo dicho dispositivo el encargado de notificar al procesador mediante la línea de interrupción que ya está preparado para continuar/terminar la operación de E/S.

## Excepciones

Son aquellas que se producen de forma síncrona a la ejecución del procesador y por tanto podrían predecirse si se analiza con detenimiento la traza del programa que en ese momento estaba siendo ejecutado en la CPU. Normalmente son causadas al realizarse operaciones no permitidas tales como la división entre 0, el desbordamiento, el acceso a una posición de memoria no permitida, etc.

Las excepciones son un tipo de interrupción sincrónica típicamente causada por una condición de error en un programa, como por ejemplo una división entre 0 o un acceso inválido a memoria en un proceso de usuario. Normalmente genera un cambio de contexto a modo supervisor para que el sistema operativo atienda el error. Así pues, las excepciones son un mecanismo de protección que permite garantizar la integridad de los datos almacenados tanto en el espacio de usuario como en el espacio kernel. Cuando el Sistema Operativo detecta una excepción intenta solucionarla, pero en caso de no poder simplemente notificará la condición de error a la aplicación/usuario y abortará la misma.

## Interrupciones por software

Las interrupciones por software son aquellas generadas por un programa en ejecución. Para generarlas, existen distintas instrucciones en el código máquina que permiten al programador producir una interrupción, las cuales suelen tener nemotécnicos tales como INT (por ejemplo, en DOS se realiza la instrucción INT 0x21 y en Unix se utiliza INT 0x80 para hacer llamadas de sistema).

La interrupción por software, también denominadas llamadas al sistema, son aquellas generadas por un programa mientras este está ejecutándose. En general, actúan de la siguiente manera:

- Un programa que se venía ejecutando luego de su instrucción I5 , llama al Sistema Operativo, por ejemplo para leer un archivo de disco (cuando un programa necesita un dato exterior , se detiene y pasa a cumplir con las tareas de recoger ese dato).

- A tal efecto, luego de I5 existe en el programa, la instrucción de código de máquina CD21, simbolizada INT 21 en Assembler, que realiza el requerimiento del paso 1. Puesto que no puede seguir la ejecución de la instrucción I6 y siguientes del programa hasta que no se haya leído el disco y esté en memoria principal dicho archivo, virtualmente el programa se ha interrumpido, siendo, además, que luego de INT 21, las instrucciones que se ejecutarán no serán del programa, sino del Sistema Operativo. Se detiene el programa y ordena en este caso mediante INT21 (interrupción predefinida) que recoge el dato solicitado, para poder seguir el programa que la ordenó).
- La ejecución de INT 21 permite hallar la subrutina del Sistema Operativo.
- Se ejecuta la subrutina del Sistema Operativo que prepara la lectura del disco.
- Luego de ejecutarse la subrutina del Sistema Operativo, y una vez que se haya leído el disco y verificado que la lectura es correcta, el Sistema Operativo ordenará reanudar la ejecución del programa autointerrumpido en espera.
- La ejecución del programa se reanuda.

# Enmascaramiento y Prioridades de las interrupciones

## a).- Enmascaramiento de las interrupciones

No siempre es posible la atención inmediata de una interrupción, siendo necesario en algunos casos posponer la atención de la misma o inhibir el tratamiento correspondiente.
En los sistemas operativos se presentan situaciones en las cuales es necesario posponer la atención de una interrupción, o bien no llevarla a cabo. Por ejemplo, si ha ocurrido una interrupción por finalización de una operación en un dispositivo de entrada/salida, y luego se produce una nueva interrupción desde otro dispositivo sin que haya terminado de atender la primera interrupción, es conveniente terminar la atención de la primera y posponer mientras tanto las acciones necesarias para atender la segunda interrupción. En otros casos, algunas interrupciones de falla de máquina no pueden ser pospuestas cuando ocurren.
El enmascaramiento de interrupciones se hace a través de los componentes de hardware, éstos pueden tomar un estado en el cual, cuando ocurra una interrupción de cierto tipo, se mantiene dicha condición como una señal, pero el mecanismo de interrupción no las tomará en cuenta hasta que no llegue el momento apropiado. En el caso de enmascaramiento total de interrupciones, es decir, que no se atienden cuando ocurren, el enmascaramiento puede ser temporal ya que el sistema operativo deshabilita la atención de las interrupciones y las habilita posteriormente.

## b).- Prioridades de las Interrupciones

En las secciones anteriores se estudiaron los diferentes tipos de interrupciones, y se explicó que la ocurrencia de éstas pueden ser síncronas (generadas por otro programa) y asíncronas (generadas por un agente externo). Es posible que durante el tratamiento de una interrupción ocurra una segunda interrupción, para lo cual podemos usar un mecanismo de enmascaramiento que retarde la atención de la segunda interrupción. Sin embargo, hay dos casos que conviene estudiar:

- La ocurrencia simultánea de varias interrupciones: En este caso es necesario establecer un mecanismo, que permita atender todas las interrupciones presentes en un determinado orden. Este orden de atención no puede ser dado por el tiempo, ya que todas ocurrieron en el mismo momento.
- Dos interrupciones consecutivas: Ocurre la segunda interrupción, sin que se haya terminado de atender la primera, pero la segunda interrupción no se puede enmascarar, es decir, debe ser

atendida de inmediato. Entonces se deberá suspender temporalmente las acciones que se llevan a cabo para el tratamiento de la primera interrupción, y proceder a tratar la segunda interrupción. Esta situación es posible, cuando la condición por la cual se presentó la segunda interrupción, puede tener efectos generales sobre el sistema completo e inclusive sobre la interrupción que se atiende.

Las dos situaciones presentadas anteriormente, implican que deben asignarse prioridades a los diferentes tipos de interrupciones, esta asignación irá en función de la importancia que representan las mismas. Además debe disponerse de un mecanismo de hardware que garantice el tratamiento de las interrupciones de mayor prioridad frente delante de aquellas de menor prioridad. Esto es posible de realizar, si se utiliza el mecanismo de enmascaramiento, de manera que cuando se esté tratando una interrupción de cierto nivel de prioridad, sean enmascaradas todas las interrupciones con prioridad igual o menor

## Mecanismos para el Tratamiento de Interrupciones

El núcleo del sistema operativo debe proveer las rutinas especiales para el manejo de interrupciones, estas rutinas deben ser capaces de reconocer el tipo específico de interrupción que ha ocurrido, para hacer el tratamiento correspondiente.

A continuación se listan el conjunto de pasos para resolver una interrupción:

1. Al momento de producirse una interrupción, debe quedar almacenada en un registro especial, la identificación del tipo de interrupción que se ha presentado.
2. Se debe salvar el estado que tenía el CPU cuando se produjo la interrupción, es decir salvar el contador de programa y los registros e indicadores en un área reservada de memoria.
3. Se hace una transferencia incondicional al punto de entrada de la rutina manejadora de interrupciones.

Los pasos 1 al 3 son ejecutados automáticamente por el mecanismo de hardware encargado de manejar las interrupciones. La dirección de comienzo de la rutina manejadora de interrupciones debe ser fija en memoria.

4. Una vez en la rutina manejadora de interrupciones, se determinará el origen de la interrupción, utilizando la información almacenada en el registro especial mencionado en el paso 1. Identificada la condición de interrupción presente, se procederá a tomar las acciones correspondientes (en general, se hará una transferencia a la rutina de servicio de interrupción que corresponde al tipo de interrupción presentado).
5. Se mantienen enmascaradas las interrupciones con prioridad igual o menor que aquella que se está procesando.
6. Al finalizar de ejecutarse la rutina manejadora de interrupciones, eventualmente continuará ejecutándose el programa que estaba corriendo cuando se produjo la interrupción, restableciendo los valores del contador de programa, registros e indicadores, desde el área de memoria donde habían sido salvados cuando se presentó la interrupción. Si hubiese interrupciones pendientes por atender, estas deberán ser tratadas, antes de dar control nuevamente al programa que inicialmente fue interrumpido.

# 13. Concurrencia y secuenciabilidad.

La concurrencia es el punto clave de los tres campos anteriores y fundamentales para el diseño de sistemas operativos. La concurrencia comprende un gran número de cuestiones de diseño, incluyendo la comunicación entre procesos, compartición y competencia por los recursos, sincronización de la ejecución de varios procesos y asignación del tiempo de procesador a los procesos. Se verá que estas cuestiones no sólo surgen en entornos de multiprocesadores y proceso distribuido, sino incluso en sistemas multiprogramados con un solo procesador.
La concurrencia puede presentarse en tres contextos diferentes:

- **Varias aplicaciones:** La multiprogramación se creó para permitir que el tiempo de procesador de la máquina fuese compartido dinámicamente entre varios trabajos o aplicaciones activas.
- **Aplicaciones estructuradas:** Como ampliación de los principios del diseño modular y la programación estructurada, algunas aplicaciones pueden implementarse eficazmente como un conjunto de procesos concurrentes.
- **Estructura del sistema operativo:** Las mismas ventajas de estructuración son aplicables a los programadores de sistemas y se ha comprobado que algunos sistemas operativos están implementados como un conjunto de procesos.

En un sistema multiprogramado con un único procesador, los procesos se intercalan en el tiempo para dar la apariencia de ejecución simultánea. Aunque no se consigue un proceso paralelo real y aunque se produce una cierta sobrecarga en los intercambios de procesos de un sitio a otro, la ejecución intercalada produce beneficios importantes en la eficiencia del procesamiento y en la estructuración de los programas. En un sistema con varios procesadores, no solo es posible intercalar los procesos, sino también superponerlos.
A primera vista, podría parecer que la intercalación y la superposición representan formas de ejecución muy diferentes y que introducen problemas distintos. De hecho, ambas técnicas pueden contemplarse como ejemplos de proceso concurrente y ambas plantean los mismos problemas. En el caso de un sistema monoprocesador, los problemas creados por la multiprogramación parten del hecho de que la velocidad relativa de ejecución de los procesos no puede predecirse. Depende de la actividad de otros procesos, de la forma en que el sistema operativo trata las interrupciones y de las políticas de planificación.

## Exclusión mutua de secciones críticas.

Cualquier servicio o capacidad que dé soporte para la exclusión mutua debe cumplir los requisitos siguientes:

1. **Debe cumplirse la exclusión mutua:** Sólo un proceso, de entre todos los que poseen secciones críticas por el mismo recurso u objeto compartido, debe tener permiso para entrar en ella en un instante dado.
2. Un proceso que se interrumpe en una sección no crítica debe hacerlo sin estorbar a los otros procesos.
3. Un proceso no debe poder solicitar acceso a una sección crítica para después ser demorado indefinidamente; no puede permitirse el interbloqueo o la inanición.

4. Cuando ningún proceso está en su sección crítica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin dilación.
5. No se pueden hacer suposiciones sobre la velocidad relativa de los procesos o su número.
6. Un proceso permanece en su sección crítica solo por un tiempo finito.

Hay varias formas de satisfacer los requisitos de exclusión mutua. Una manera es dejar la responsabilidad a los procesos que deseen ejecutar concurrentemente. Así pues, tanto si son programas del sistema como de aplicación, los procesos deben coordinarse unos con otros para cumplir la exclusión mutua, sin ayuda por parte del lenguaje de programación o del sistema operativo. Estos métodos se conocen como soluciones por software. Aunque las soluciones por software son propensas a errores y a una fuerte carga de proceso, resulta útil estudiar estos métodos para tener un mejor entendimiento de la complejidad del proceso concurrente. Un segundo método propone el uso de instrucciones de la máquina a tal efecto. Estas tienen la ventaja de reducir la sobrecarga. El tercer método consiste en dar algún tipo de soporte en el sistema operativo.

- **Algoritmo de Dekker**

Dijkstra [DIJ65] presentó un algoritmo de exclusión mutua para dos procesos que diseñó el matemático holandés Dekker. Según Dijkstra, la solución se desarrolla por etapas. Este método tiene la ventaja de ilustrar la mayoría de los errores habituales que se producen en la construcción de programas concurrentes. A medida que se construya el algoritmo, se emplearán algunas ilustraciones pintorescas tomadas de Ben-Ari.

- **Algoritmo de Peterson**

El algoritmo de Dekker resuelve el problema de la exclusión mutua pero con un programa complejo, difícil de seguir y cuya corrección es difícil de demostrar. Peterson [PETE81] desarrollado una solución simple y elegante. Como antes, la variable global señal indica La posición de cada proceso con respecto a la exclusión mutua y la variable global turno resuelve los conflictos de simultaneidad.

Se puede demostrar fácilmente que se cumple La exclusión mutua. Considérese el proceso P0. Una vez que ha puesto señal [0] a cierto, P1 no puede entrar en su sección critica. Si P1 está aún en su sección critica, señal [1] = cierto y P0 está bloqueado para entrar en su sección crítica. Por otro lado, se impide el bloqueo mutuo. Supóngase que P0 está bloqueado en su bucle **while.** Esto significa que señal es cierto y turno = 1. P0 puede entrar en su sección crítica cuando señal [1] se ponga a falso o cuando turno se ponga a 0. Considérense ahora los siguientes casos exhaustivos:

1. P1 no está interesado en entrar en su sección crítica. Este caso es imposible porque implica que señal [l]= falso.

2. P1 está esperando entrar en su sección crítica. Este caso es también imposible porque si turno = 1, P1 podrá entrar en su sección critica.
3. P1 entra en su sección crítica varias veces y monopoliza el acceso a ella. Esto no puede pasar porque P1 está obligado a dar a P0 una oportunidad poniendo turno a 0 antes de cada intento de entrar en su sección crítica.

Así pues, se tiene una solución simple al problema de la exclusión mutua para dos procesos. Es más, el algoritmo de Peterson se puede generalizar fácilmente al caso de n procesos

# Sincronización de procesos en S.C.

# Mecanismo de semáforos.

El primer gran avance en la resolución de los problemas de procesos concurrentes llegó en 1965, con el tratado de Dijkstra sobre la cooperación entre procesos secuénciales [DIJK65] Dijkstra estaba interesado en el diseño de un sistema operativo como un conjunto de proce**sos** secuénciales cooperantes y en el desarrollo de mecanismos eficientes y fiables para dar soporte a la cooperación. Los procesos de usuario podrán utilizar estos mecanismos en tanto que el procesador y el sistema operativo los hagan disponibles.

El principio fundamental es el siguiente: Dos o más procesos pueden cooperar por medio de simples señales, de forma que se pueda obligar a detenerse a un proceso en una posición determinada hasta que reciba una señal específica. Cualquier requisito complicado de coordinación puede satisfacerse por medio de la estructura de señales adecuada. Para la señalización, se usan variables especiales llamados semáforos. Para transmitir una señal por el Semáforo s, los procesos ejecutan la primitiva signal(s). Para recibir una señal del semáforo s, los procesos ejecutan la primitiva wait(s); si la señal correspondiente aún no se ha transmitido, el proceso es suspendido hasta que tenga lugar la transmisión .

Para lograr el efecto deseado, se pueden contemplar los semáforos como variables que tienen un valor entero sobre el que se definen las tres operaciones siguientes:

1. Un semáforo puede inicializarse con un valor no negativo.
2. La operación wait decrementa el valor del semáforo. Si el valor se hace negativo, el proceso que ejecuta el wait se bloquea.
3. La operación signal incrementa el valor del semáforo. Si el valor no es positivo, se desbloquea a un proceso bloqueado por una operación wait.

Aparte de estas tres operaciones, no hay forma de examinar o manipular los semáforos.
Las primitivas wait y signal se suponen atómicas, es decir, no pueden ser interrumpidas y cada rutina puede considerarse como un paso indivisible. En principio, los semáforos binarios son más sencillos de implementar y puede demostrarse que tienen la misma potencia de expresión que los semáforos generales Tanto en los semáforos como en los semáforos binarios se emplea una cola para mantener

los procesos esperando en el semáforo. La definición no dicta el orden en el que se quitan los procesos de dicha cola. La política más equitativa es la FIFO: el proceso que ha estado bloqueado durante más tiempo se libera de la cola. La única exigencia estricta es que un proceso no debe quedar retenido en la cola de un semáforo indefinidamente porque otros procesos tengan preferencia.

## Mecanismo de monitores.

A fin de facilitar la escritura de programas correctos, Hoare (1974) y Brinch Hansen (1975) propusieron una primitiva de sincronización de nivel más alto llamada monitor. Sus propuestas tenían pequeñas diferencias, que describiremos más adelante. Un monitor es una colección de procedimientos, variables y estructuras de datos que se agrupan en un tipo especial de módulo o paquete. Los procesos pueden invocar los procedimientos de un monitor en el momento en que deseen, pero no pueden acceder directamente a las estructuras de datos internas del monitor desde procedimientos declarados afuera del monitor.
Los monitores poseen una propiedad especial que los hace útiles para lograr la exclusión mutua: sólo un proceso puede estar activo en un monitor en un momento dado. Los monitores son una construcción de lenguaje de programación, así que el compilador sabe que son especiales y puede manejar las llamadas a procedimientos de monitor de una forma diferente a como maneja otras llamadas a procedimientos. Por lo regular, cuando un proceso invoca un procedimiento de monitor, las primeras instrucciones del procedimiento verifican si hay algún otro proceso activo en ese momento dentro del monitor. Si así es, el proceso invocador se suspende hasta que el otro proceso abandona el monitor. Si ningún otro proceso está usando el monitor, el proceso invocador puede entrar.
Es responsabilidad del compilador implementar la exclusión mutua en las entradas a monitores, pero una forma común es usar un semáforo binario. Puesto que el compilador, no el programador, se está encargando de la exclusión mutua, es mucho menos probable que algo salga mal. En cualquier caso, la persona que escribe el monitor no tiene que saber cómo el compilador logra la exclusión mutua; le basta con saber que si convierte todas las regiones críticas en procedimientos de monitor, dos procesos nunca podrán ejecutar sus regiones críticas al mismo tiempo.
Aunque los monitores ofrecen una forma fácil de lograr la exclusión mutua, esto no es suficiente, como acabamos de ver. También necesitamos un mecanismo para que los procesos se bloqueen cuando no puedan continuar. En el problema de productor-consumidor, es fácil colocar todas las pruebas para determinar si el buffer está lleno o está vacío en procedimientos de monitor, pero ¿cómo deberá bloquearse el productor cuando encuentra lleno el buffer?

## Interbloqueo (DeadLock).

Es el bloqueo permanente de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros. A diferencia de otros problemas de la gestión concurrente de procesos, para el caso general no existe una solución eficiente. En esta sección, se examinará la naturaleza del problema del interbloqueo.

Todos los interbloqueos suponen demandas contradictorias de recursos por parte de dos o más procesos. Los dos ejes del diagrama representan el avance de los dos procesos en términos de instrucciones ejecutadas. El avance conjunto de los dos procesos se representa entonces con una secuencia discreta

de puntos en el espacio. Las líneas horizontales o verticales representan el intervalo de tiempo en el que sólo uno de los procesos está ejecutándose (intercalado); una línea diagonal significa ejecución simultánea (solapamiento). Supóngase que existe un punto en la ejecución de cada proceso en el que se requiere el uso exclusivo de ambos recursos, Rl y R2, para continuar. En el ejemplo, llega un punto en el que el proceso Pl ha adquirido el recurso Rl y el proceso P2 ha adquirido el recurso R2 y cada proceso necesita el otro recurso. Este es el punto de interbloqueo.
El interbloqueo se produce si cada proceso retiene un recurso y solicita el otro. Puede parecer que es un error de programación en lugar de un error del diseño del sistema operativo. Sin embargo, se ha visto que el diseño de un programa concurrente entraña gran dificultad. Se producen interbloqueos como éste y la causa está frecuentemente en la compleja lógica del programa, lo que hace más difícil su detección. Una posible estrategia para resolver estos interbloqueos es imponer restricciones en el diseño del sistema sobre el orden en el que se solicitan los recursos.

## Prevención.

La estrategia de prevención del interbloqueo consiste, a grandes rasgos, en diseñar un sistema de manera que esté excluida, a priori, la posibilidad de interbloqueo. Los métodos para prevenir el interbloqueo son de dos tipos. Los métodos indirectos consisten en impedir la aparición de alguna de las tres condiciones necesarias, antes mencionadas (condiciones 1 a 3). Los métodos directos consisten en evitar la aparición del círculo vicioso de espera (condición 4). Se examinarán a continuación las técnicas relacionadas con cada una de las cuatro condiciones. Exclusión Mutua
En general, la primera de las cuatro condiciones no puede anularse. Si el acceso a un recurso necesita exclusión mutua, el sistema operativo debe soportar la exclusión mutua. Algunos recursos, como los archivos, pueden permitir varios accesos para lectura, pero sólo accesos exclusivos para escritura. Incluso en este caso, se puede producir interbloqueo si más de un proceso necesita permiso de escritura.

### Retención y Espera

La condición de retención y espera puede prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. Esta solución resulta ineficiente por dos factores. En primer lugar, un proceso puede estar suspendido durante mucho tiempo, esperando que se concedan todas sus solicitudes de recursos, cuando de hecho podría haber avanzado con sólo algunos de los recursos. Y en segundo lugar, los recursos asignados a un proceso pueden permanecer sin usarse durante periodos considerables, tiempo durante el cual se priva del acceso a otros procesos.

### No apropiación

La condición de no apropiación puede prevenirse de varias formas. Primero, si a un proceso que retiene ciertos recursos se le deniega una nueva solicitud, dicho proceso deberá liberar sus recursos anteriores y solicitarlos de nuevo, cuando sea necesario, junto con el recurso adicional. Por otra parte, si un proceso solicita un recurso que actualmente está retenido por otro proceso, el sistema operativo

puede expulsar al segundo proceso y exigirle que libere sus recursos. Este último esquema evitará el interbloqueo sólo si no hay dos procesos que posean la misma prioridad.

Esta técnica es práctica sólo cuando se aplica a recursos cuyo estado puede salvarse y restaurarse más tarde de una forma fácil, como es el caso de un procesador.

## Círculo Vicioso de Espera

La condición del círculo vicioso de espera puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Si a un proceso se le han asignado recursos de tipo R, entonces sólo podrá realizar peticiones posteriores sobre los recursos de los tipos siguientes a R en la ordenación.
Para comprobar el funcionamiento de esta estrategia, se asocia un índice a cada tipo de recurso. En tal caso, el recurso R, antecede a R, en la ordenación si i <j. Entonces, supón gase que dos procesos A y B se interbloquean, porque A ha adquirido R, y solicitado Ry, mientras que B ha adquirido R;y solicitado R¿. Esta situación es imposible porque implica que í<jyj<i.
Como en la retención y espera, la prevención del círculo vicioso de espera puede ser ineficiente, retardando procesos y denegando accesos a recursos innecesariamente.

## Detección.

Las estrategias de prevención del interbloqueo son muy conservadoras; solucionan el pro-blema del interbloqueo limitando el acceso a los recursos e imponiendo restricciones a los procesos. En el lado opuesto, las estrategias de detección del interbloqueo no limitan el acceso a los recursos ni restringen las acciones de los procesos. Con detección del interbloqueo, se concederán los recursos que los procesos necesiten siempre que sea posible. Periódicamente, el sistema operativo ejecuta un algoritmo que permite detectar la condición de círculo vicioso de espera descrita en el punto 4 anterior. Puede emplearse cualquier algoritmo de detección de ciclos en grafos dirigidos.

El control del interbloqueo puede llevarse a cabo tan frecuentemente como las solicitudes de recursos o con una frecuencia menor, dependiendo de la probabilidad de que se produzca el interbloqueo. La comprobación en cada solicitud de recurso tiene dos ventajas: Conduce a una pronta detección y el algoritmo es relativamente simple, puesto que está basado en cambios increméntales del estado del sistema. Por otro lado, tal frecuencia de comprobaciones consume un tiempo de procesador considerable.
Una vez detectado el interbloqueo, hace falta alguna estrategia de recuperación. Las técnicas siguientes son posibles enfoques, enumeradas en orden creciente de sofisticación:

1. Abandonar todos los procesos bloqueados. Esta es, se crea o no, una de las soluciones más comunes, si no la más común, de las adoptadas en un sistema operativo.
2. Retroceder cada proceso interbloqueado hasta algún punto de control definido previamente y volver a ejecutar todos los procesos. Es necesario que haya disponibles unos mecanismos de retroceso y reinicio en el sistema. El riesgo de esta solución radica en que puede repetirse el interbloqueo original. Sin embargo, el no determinismo del procesamiento concurrente asegura, en general, que esto no va a pasar.
3. Abandonar sucesivamente los procesos bloqueados hasta que deje de haber interbloqueo. El orden en el que se seleccionan los procesos a abandonar seguirá un criterio de mínimo coste. Después de abandonar cada proceso, se debe ejecutar de nuevo el algoritmo de detección para ver si todavía existe interbloqueo.

4. Apropiarse de recursos sucesivamente hasta que deje de haber interbloqueo. Como en el punto 3, se debe emplear una selección basada en coste y hay que ejecutar de nuevo el algoritmo de detección después de cada apropiación. Un proceso que pierde un recurso por apropiación debe retroceder hasta un momento anterior a la adquisición de ese recurso.

Para los puntos 3 y 4, el criterio de selección podría ser uno de los siguientes, consistentes en escoger el proceso con:

- La menor cantidad de tiempo de procesador consumido hasta ahora
- El menor número de líneas de salida producidas hasta ahora
- El mayor tiempo restante estimado
- El menor número total de recursos asignados hasta ahora
- La prioridad más baja

Algunas de estas cantidades son más fáciles de medir que otras. El tiempo restante estimado deja lugar a dudas, especialmente. Además, aparte de las medidas de prioridad, no existe otra indicación del "coste" para el usuario frente al coste para el sistema en conjunto.

## Recuperación.

Un sistema que pretenda recuperarse del interbloqueo, debe invocar a un algoritmo de detección cuando lo considere oportuno (ej. periódicamente)

Formas de intentar la recuperación:

- Terminación de procesos
- matando a todos los procesos implicados (drástico)
- matando a uno de los procesos ¿cuál? el que más recursos libere
- el que menos tiempo lleve en ejecución…
- Retrocediendo la ejecución de algún proceso (*rollback*) muy complicado de implementar y necesita que el programa esté diseñado para que pueda retroceder

Expropiación de recursos

Selección de la víctima

¿Qué recursos y de qué procesos se expropian ?

En ambos casos (terminación de procesos o expropiación de recursos) hay que tener cuidado de no provocar la inanición de procesos

# Niveles, objetivos y criterios de planificación.

Se consideran tres niveles importantes de planificación, los que se detallan a continuación:

- **Planificación de alto nivel:** Se encarga de llevar procesos de disco a memoria y viceversa. Seleccionando los trabajos que deben admitirse en el sistema.También se denomina Planificación de trabajos. Determina a qué trabajos se les va a permitir competir activamente por los recursos del sistema, lo cual se denomina Planificación de admisión. Administrar todos los recursos del sistema excepto el CPU. Mantiene las colas de procesos bloqueados y suspendidos. Controla la creación de procesos. Maneja el nivel de multiprogramación.
- **Planificación de nivel intermedio:** En algunos casos, en especial cuando el sistema está sobrecargado, el planificador de nivel medio encuentra ventajoso retirar trabajos activos de la memoria para reducir el grado de multiprogramación, y por lo tanto, permitir que los trabajos se completen mas aprisa. Este subadministrador controla los trabajos que se intercambian hacia fuera y de regreso. Determina a qué procesos se les puede permitir competir por la cpu. Efectúa “suspensiones” y “activaciones” (“reanudaciones”) de procesos. Debe ayudar a alcanzar ciertas metas en el rendimiento total del sistema. Equilibrar la administración de trabajos en el sistema con la asignación del CPU a dichos procesos. Nivelar la carga del sistema (procesos activos y pasivos).
- **Planificación de bajo nivel:** Se encarga de pasar de un proceso a otro en memoria principal. Determinando a cuál proceso listo se le asignará el CPU cuando éste se encuentra disponible. Determina a qué proceso listo se le asigna la cpu cuando esta queda disponible y asigna la cpu al mismo, es decir que “despacha” la cpu al proceso.

# OBJETIVOS DE PLANIFICACIÓN

Los objetivos de la planificación del procesador son los siguientes e involucran a los conceptos detallados seguidamente:

- Ser justa: Todos los procesos son tratados de igual manera. Ningún proceso es postergado indefinidamente.
- Maximizar la capacidad de ejecución: Maximizar el número de procesos servidos por unidad de tiempo.
- Maximizar el número de usuarios interactivos que reciban unos tiempos de respuesta aceptables: En un máximo de unos segundos.
- Ser predecible: Un trabajo dado debe ejecutarse aproximadamente en la misma cantidad de tiempo independientemente de la carga del sistema.
- Minimizar la sobrecarga: No suele considerarse un objetivo muy importante.
- Equilibrar el uso de recursos: Favorecer a los procesos que utilizarán recursos infrautilizados.
- Equilibrar respuesta y utilización: La mejor manera de garantizar buenos tiempos de respuesta es disponer de los recursos suficientes cuando se necesitan, pero la utilización total de recursos podrá ser pobre.
- Evitar la postergación indefinida: Se utiliza la estrategia del “envejecimiento” . Mientras un proceso espera por un recurso su prioridad debe aumentar, así la prioridad llegará a ser tan alta que el proceso recibirá el recurso esperado.
- Asegurar la prioridad: Los mecanismos de planificación deben favorecer a los procesos con prioridades más altas.
- Dar preferencia a los procesos que mantienen recursos claves: Un proceso de baja prioridad podría mantener un recurso clave, que puede ser requerido por un proceso de más alta prioridad. Si el recurso es no apropiativo,

el mecanismo de planificación debe otorgar al proceso un tratamiento mejor del que le correspondería normalmente, puesto que es necesario liberar rápidamente el recurso clave.

- Dar mejor tratamiento a los procesos que muestren un "comportamiento deseable": Un ejemplo de comportamiento deseable es una tasa baja de paginación.
- Degradarse suavemente con cargas pesadas: Un mecanismo de planificación no debe colapsar con el peso de una exigente carga del sistema. Se debe evitar una carga excesiva mediante las siguientes acciones: No permitiendo que se creen nuevos procesos cuando la carga ya es pesada. Dando servicio a la carga más pesada al proporcionar un nivel moderadamente reducido de servicio a todos los procesos.

## CRITERIOS DE PLANIFICACIÓN

- Equidad Garantizar que cada proceso obtiene su proporción justa de la cpu.
- Eficacia Mantener ocupada la cpu el ciento por ciento del tiempo.
- Tiempo de respuesta Minimizar el tiempo de respuesta para los usuarios interactivos.
- Tiempo de regreso Minimizar el tiempo que deben esperar los usuarios por lotes (batch) para obtener sus resultados.
- Rendimiento Maximizar el número de tareas procesadas por hora.

# Técnicas de administración del planificador.

## FIFO

Cuando se tiene que elegir a qué proceso asignar la CPU se escoge al que llevará más tiempo listo. El proceso se mantiene en la CPU hasta que se bloquea voluntariamente.

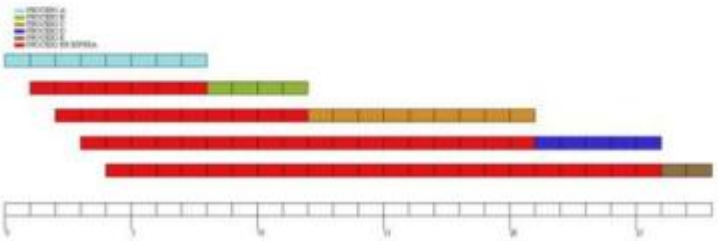
La ventaja de este algoritmo es su fácil implementación, sin embargo, no es válido para entornos interactivos ya que un proceso de mucho cálculo de CPU hace aumentar el tiempo de espera de los demás procesos . Para implementar el algoritmo (ver figura 2) sólo se necesita mantener una cola con los procesos listos ordenada por tiempo de llegada. Cuando un proceso pasa de bloqueado a listo se sitúa el último de la cola.

En esta política de planificación, el procesador ejecuta cada proceso hasta que termina, por tanto, los procesos que en cola de procesos preparados permanecerán encolados en el orden en que lleguen hasta que les toque su ejecución. Este método se conoce también como FIFO (fist input, first output,Primero en llegar primero en salir).
Se trata de una política muy simple y sencilla de llevar a la práctica, pero muy pobre en cuanto a su comportamiento.
La cantidad de tiempo de espera de cada proceso depende del número de procesos que se encuentren en la cola en el momento de su petición de ejecución y del tiempo que cada uno de ellos tenga en uso al procesador, y es independiente de las necesidades del propio proceso.
Sus características son:

- No apropiativa.
- Es justa, aunque los procesos largos hacen esperar mucho a los cortos.
- Predecible.
- El tiempo medio de servicio es muy variable en función del número de procesos y su duración.

Ejemplo :
Proceso A → Tiempo ejecución → Tiempo llegada → Tiempo finaliza → Tiempo retorno → Tiempo espera
.

| Proceso | Tiempo de ejecución | Tiempo de llegada | Tiempo de comienzo | Tiempo de finalización | Tiempo de retorno | Tiempo de espera |
|---|---|---|---|---|---|---|
| A | 8 | 0 | 0 | 8 | 8 | 0 |
| B | 4 | 1 | 9 | 12 | 12-1 = 11 | 11 – 4 = 7 |
| C | 9 | 2 | 13 | 21 | 21-2 = 19 | 19 - 9 =10 |
| D | 5 | 3 | 21 | 26 | 26-3 = 23 | 23 – 5 = 18 |
| E | 2 | 4 | 26 | 28 | 28-4 = 24 | 24 -2 = 22 |



En a) el proceso P7 ocupa la CPU, los procesos P2, P4 y P8 se mantienen en la lista de preparados. En b) P7 se bloquea (ya sea al realizar una E/S, una operación WAIT sobre un semáforo a cero u otra causa) y P2 pasa a ocupar la CPU. En c) ocurre un evento (finalización de la operación de E/S, operación SIGNAL, ...) que desbloquea a P7, esto lo vuelve listo, pasando al final de la cola de procesos listos.

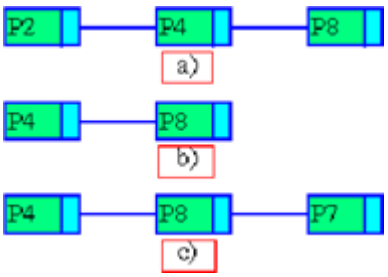



Figura 6.Lista de procesos preparados en FIFO.

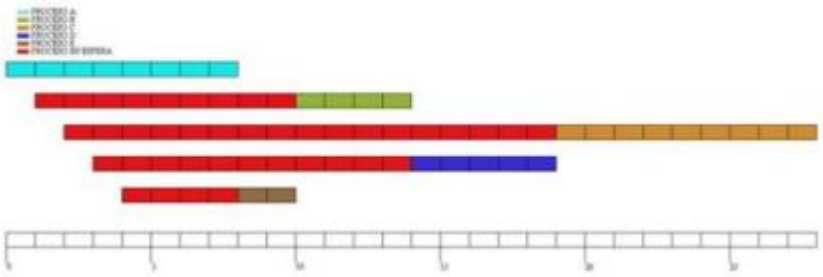
## SJF

Al igual que en el algoritmo FIFO las ráfagas se ejecutan sin interrupción, por tanto, sólo es útil para entornos batch. Su característica es que cuando se activa el planificador, éste elige la ráfaga de menor duración. Es decir, introduce una noción de prioridad entre ráfagas. Hay que recordar que en los entornos batch se pueden hacer estimaciones del tiempo de ejecución de los procesos.

En este algoritmo , da bastante prioridad a los procesos más cortos a la hora de ejecución y los coloca en la cola.
Ejemplo:
Una cola de personas en Mercadona delante de la caja , la persona que menos compra lleva esa pasa primero.

SJF

| Proceso | Tiempo de ejecución | Tiempo de llegada | Tiempo de comienzo | Tiempo de finalización | Tiempo de retorno | Tiempo de espera |
|---|---|---|---|---|---|---|
| A | 8 | 0 | 0 | 8 | 8 | 0 |
| B | 4 | 1 | 10 | 14 | 14 - 1= 13 | 13 – 4 = 9 |
| C | 9 | 2 | 19 | 28 | 28 -2 = 26 | 26 – 9 = 17 |
| D | 5 | 3 | 14 | 19 | 19 – 3 = 16 | 16 – 5 = 11 |
| E | 2 | 4 | 8 | 10 | 10 – 4= 6 | 6 – 2 = 4 |



En resumen, este algoritmo selecciona al proceso con el próximo tiempo ejecución más corto. en proceso corto saltará a la cabeza de la cola. Ejecución de un proceso consiste en ciclos de ejecución de CP y ciclos de espera por E/S. El algoritmo selecciona aquel proceso cuyo próximo ciclo de ejecución de CP sea menor. El problema está en conocer dichos valores, pero podemos predecirlos usando la información de los ciclos anteriores ejecutados

La ventaja que presenta este algoritmo sobre el algoritmo FIFO es que minimiza el tiempo de finalización promedio, como puede verse en el siguiente ejemplo:
Ej: Supongamos que en un momento dado existen tres ráfagas listos R1, R2 y R3, sus tiempos de ejecución respectivos son 24, 3 y 3 ms. El proceso al que pertenece la ráfaga R1 es la que lleva más tiempo ejecutable, seguido del proceso al que pertenece R2 y del de R3. Veamos el tiempo medio de finalización (F) de las ráfagas aplicando FIFO y SJF:

FIFO F = (24 + 27 + 30) / 3 = 27 ms.
SJF F = (3 + 6 + 30) / 3 = 13 ms.

Se puede demostrar que este algoritmo es el óptimo. Para ello, consideremos el caso de cuatro ráfagas, con tiempos de ejecución de a, b, c y d. La primera ráfaga termina en el tiempo a, la segunda termina en el tiempo a+b, etc. El tiempo promedio de finalización es (4a+3b+2c+d)/4. Es evidente que a contribuye más al promedio que los demás tiempos, por lo que debe ser la ráfaga más corta, b la siguiente, y así sucesivamente. El mismo razonamiento se aplica a un número arbitrario de ráfagas.

## Round Robin

Este es uno de los algoritmos más antiguos, sencillos y equitativos en el reparto de la CPU entre los procesos, muy válido para entornos de tiempo compartido. Cada proceso tiene asignado un intervalo de tiempo de ejecución, llamado cuantum o cuanto. Si el proceso agota su cuantum de tiempo, se elige a otro proceso para ocupar la CPU. Si el proceso se bloquea o termina antes de agotar su cuantum también se alterna el uso de la CPU. El round robin es muy fácil de implementar. Todo lo que necesita el planificador es mantener una lista de los procesos listos, como se muestra en la figura 6.2.
Es un método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.
Round Robin es uno de los algoritmos de planificación de procesos más complejos y difíciles, dentro de un sistema operativo asigna a cada proceso una porción de tiempo equitativa y ordenada, tratando a todos los procesos con la misma prioridad.
Se define un intervalo de tiempo denominado cuanto, cuya duración varía según el sistema. La cola de procesos se estructura como una cola circular. El planificado la recorre asignando un cuanto de tiempo a cada proceso. La organización de la cola es FIFO.

En esta figura en a) el proceso P7 ocupa la CPU. En b) P7 se bloquea pasando P2 a ocupar la CPU. En c) P2 agota su cuantum con lo que pasa al final de la lista y P4 ocupa la CPU. La figura 4 representa un ejemplo más largo de la ocupación de la CPU utilizando el algoritmo round robin.

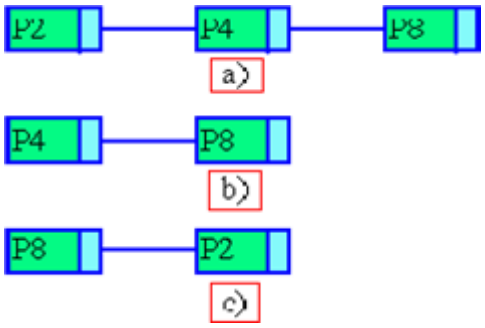
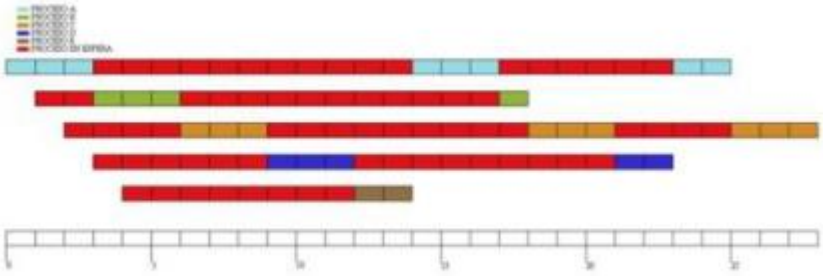



Figura 7. Lista de procesos preparados por RR.

Robin

| Proceso | Tiempo de ejecución | Tiempo de llegada | Tiempo de comienzo | Tiempo de finalización | Tiempo de retorno | Tiempo de espera |
|---|---|---|---|---|---|---|
| A | 8 | 0 | 0-14 -23 | 3 -17 -25 | 25 | 25 -8 = 17 |
| B | 4 | 1 | 3 -17 | 6 -18 | 18 - 1= 17 | 17 -4 = 13 |
| C | 9 | 2 | 6 – 18 -25 | 9 -21 -28 | 28 -2 = 26 | 26 – 9 = 17 |
| D | 5 | 3 | 9 – 21 | 12 -23 | 23 – 3 = 20 | 20 – 5 = 15 |
| E | 2 | 4 | 12 | 14 | 14 – 4= 4 | 10 – 2 = 2 |

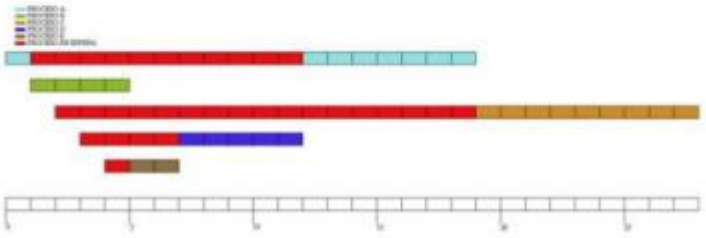


## SRTF “Short Remaining Time First”

Es similar al SJF, con la diferencia de que si un nuevo proceso pasa a listo se activa el dispatcher para ver si es más corto que lo que queda por ejecutar del proceso en ejecución. Si es así, el proceso en ejecución pasa a listo y su tiempo de estimación se decremento con el tiempo que ha estado ejecutándose.
Los procesos llegan a la cola y solicitan un intervalo de CPU
– Si dicho intervalo es inferior al que le falta al proceso en ejecución para abandonar la CPU, el nuevo proceso pasa a la CPU y el que se ejecutaba a la cola de preparados.

SRTF

| Proceso | Tiempo de ejecución | Tiempo de llegada | Tiempo de comienzo | Tiempo de finalización | Tiempo de retorno | Tiempo de espera |
|---|---|---|---|---|---|---|
| A | 8 | 0 | 0-12 | 1-19 | 19 | 19-8 = 11 |
| B | 4 | 1 | 1 | 5 | 5 - 1= 4 | 4-4 = 0 |
| C | 9 | 2 | 19 | 28 | 28 -2 = 26 | 26 – 9 = 17 |
| D | 5 | 3 | 7 | 12 | 12 – 3 = 9 | 9 – 5 = 4 |
| E | 2 | 4 | 5 | 7 | 7 – 4= 3 | 3 – 2 = 1 |



- El intervalo de CPU es difícil de predecir
- Posibilidad de inanición: los trabajos largos no se ejecutarán mientras haya trabajos cortos.

## Queves multi-level.

Un algoritmo de planificacion mediante 'colas multinivel' divide la cola de procesos preparados en varias colas distintas. Los procesos se asignan permanentemente a una cola, generalmente en función de alguna propiedad del proceso, como por ejemplo el tamaño memoria, la prioridad del proceso o el tipo de proceso. Cada cola tiene su propio algoritmo de planificación. Por ejemplo, pueden emplearse colas distintas para los procesos de primer plano y de segundo plano. La cola de primer plano puede planificarse mediante un algoritmo por turnos, mientras que para la cola de segundo plano puede emplearse un algoritmo FCFS.
Si los procesos se pueden clasificar según sus cualidades, es posible dividir la lista de procesos listos (ready queue) en varias colas (una para cada clasificación).
Los procesos son asignados permanentemente a una de las colas.
Cada cola tendrá su propio algoritmo de planificación.
Además, se debe tener una estrategia de planificación entre las diferentes colas. Por ejemplo, una cola tendrá prioridad sobre otra.

## Multi-level feedback queues.

Se diferencia con el anterior en que procesos pueden cambiar de cola (nivel). Se basa en categorizar los procesos según el uso de CPU (*CPUburst*) que tengan. La cola de mayor prioridad será la de los procesos *I/O-bound* y la de menor la de procesos con alto *CPU-bound*. De esta forma, se garantiza que los procesos con poco uso de procesador tengan mayor prioridad, y los que consumen mucho procesador tendrán baja prioridad.

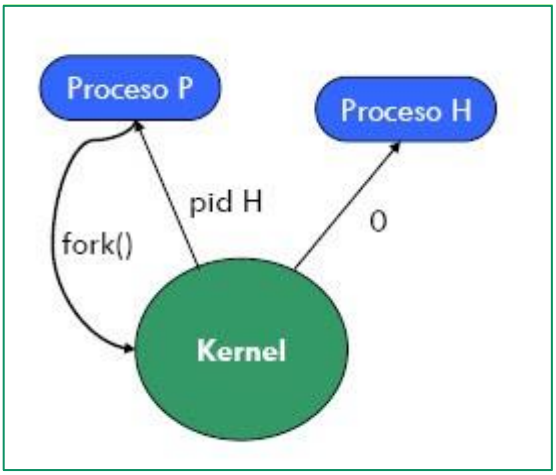
Los procesos, según el consumo de CPU que hagan, serán promovidos a una cola de mayor prioridad o rebajados a una de menor prioridad.

Un planificador Multilevel-Feedback-Queue es definido por:

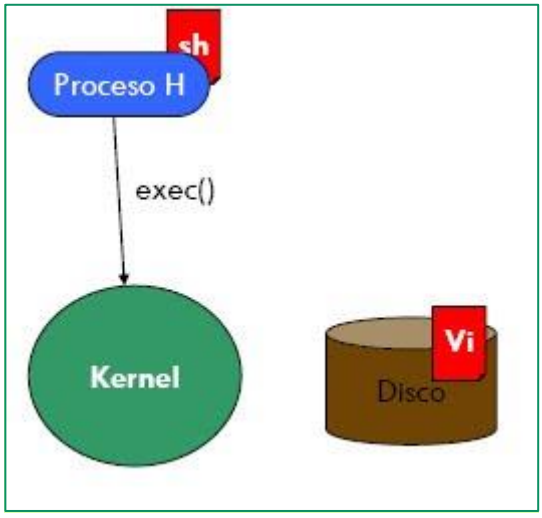
* El número de colas.
* El algoritmo de planificación para cada cola.
* El método utilizado para promover a un proceso a una cola de mayor prioridad.
* El método utilizado para bajar a un proceso a una cola de menor prioridad. * El método utilizado para determinar a que cola será asignado un proceso cuando este *pronto*.

# Creación de Procesos en Linux

- ❏ En la familia Unix se distingue entre crear procesos y ejecutar nuevos programas.
- ❏ La llamada al sistema para crear un nuevo proceso se denomina fork().
- ❏ Esta llamada crea una copia casi idéntica del proceso padre.
  - Ambos procesos, padre e hijo, continúan ejecutándose en paralelo.
  - El padre obtiene como resultado de la llamada a fork()el pid del hijo y el hijo obtiene 0.
  - Algunos recursos no se heredan (p.ej. señales pendientes).




- El proceso hijo puede invocar la llamada al sistema exec*():
  - sustituye su imagen en memoria por la de un programa diferente
- El padre puede dedicarse a crear más hijos, o esperar a que termine el hijo:
  - wait()lo saca de la cola de “listos” hasta que el hijo termina.



**IDentificadores**

PROCESS ID (PID)

Al crearse un nuevo proceso se le asigna un identificador de proceso único. Este número debe utilizarse por el administrador para referirse a un proceso dado al ejecutar un comando.

Los PID son asignados por el sistema a cada nuevo proceso en orden creciente comenzando desde cero. Si antes de un reboot del sistema se llega al nro. máximo, se vuelve a comenzar desde cero, salteando los procesos que aún estén activos.

PARENT PROCESS ID (PPID)

La creación de nuevos procesos en Unix se realiza por la vía de duplicar un proceso existente invocando al comando fork(). Al proceso original se le llama “padre” y al nuevo proceso “hijo”. El PPID de un proceso es el PID de su proceso padre.

El mecanismo de creación de nuevos procesos en Unix con el comando fork() se ve con más detalle en el apartado “Ciclo de vida de un proceso”.

# Ciclo de vida de un proceso

El mecanismo de creación de un proceso en Unix es un poco peculiar. Un proceso se crea invocando a una función del sistema operativo llamada **fork()**. La función fork() crea una copia idéntica del proceso que la invoca con excepción de:

- El nuevo proceso tiene un PID diferente
- El PPID del nuevo proceso es el PID del proceso original
- Se reinicia la información de tarificación del proceso (uso de CPU, etc.)

Al retorno de fork() se siguen ejecutando las siguientes sentencias del programa en forma concurrente. Para distinguir entre los dos procesos la función fork() devuelve un cero al proceso hijo y el PID del nuevo proceso al proceso padre. Normalmente el proceso hijo lanza luego un nuevo programa ejecutando alguna variante de comando exec(). En el recuadro puede verse un ejemplo del uso de fork.

```
kidpid = fork()
if (kidpid==0) {
    /* soy el hijo, p. ej. lanzo un nuevo prog. */
       exec(...)
  }
  else{
     /* soy el padre */
        ...
        wait()        /* espero exit() del hijo */
  }
```

Si este es el mecanismo para crear un proceso, entonces ¿quién lanza a correr el primer proceso? Luego del boot del sistema el kernel instala y deja corriendo un proceso llamado **init** con PID=1. Una de las funciones principales de init es lanzar mediante fork() intérpretes de comandos que a su vez lanzarán los scripts de inicialización del sistema y los procesos de los usuarios. Además de init el kernel lanza algunos procesos más cuyo nombre y función varía en los diferentes sabores de Unix. A excepción de estos procesos lanzados por el kernel al inicio, todos los demás son descendientes de init.

Normalmente un proceso termina invocando a la función **exit()** pasando como parámetro un código de salida o **exit code**. El destinatario de ese código de salida es el proceso padre. El proceso padre puede esperar la terminación de su proceso hijo invocando la función **wait().** Esta función manda al padre a dormir hasta que el hijo ejecute su exit() y devuelve el exit code del proceso hijo.

Cuando el proceso hijo termina antes que el padre, el kernel debe conservar el valor del exit code para pasarlo al padre cuando ejecute wait(). En esta situación se dice que el proceso hijo está en el estado **zombie**. El kernel devuelve todas las áreas de memoria solicitadas por el proceso pero debe mantener alguna información sobre el proceso (al menos su PID y el exit code).

Cuando el proceso padre termina primero el kernel encarga a **init** la tarea de ejecutar el wait() necesario para terminar todo en forma ordenada. A menudo **init** falla en esta función y suelen quedar procesos en estado zombie hasta un nuevo reboot. Dado que un proceso zombie no consume recursos fuera de su PID, esto por lo general no provoca problemas.

## Tipos de Procesos

Características:

- Un proceso consta de código, datos y pila.
- Los procesos existen en una jerarquía de árbol (varios Hijos, un sólo padre).
- El sistema asigna un identificador de proceso (PID) único al iniciar el proceso.
- El planificador de tareas asigna un tiempo compartido para el proceso según su prioridad (sólo *root* puede cambiar prioridades).

**Ejecución en 1*er* plano:**

proceso iniciado por el usuario o interactivo.

**Ejecución en 2*o* plano:**

proceso no interactivo que no necesita ser iniciado por el usuario.

**Demonio:**

proceso en 2*o* plano siempre disponible, que da servicio a varias tareas (debe ser propiedad del usuario *root*).

**Proceso zombi:**

proceso parado que queda en la tabla de procesos hasta que termine su padre. Este hecho se produce cuando el proceso padre no recoge el código de salida del proceso hijo.

**Proceso huérfano:**

proceso en ejecución cuyo padre ha finalizado. El nuevo identificador de proceso padre (PPID) coincide con el identificador del proceso **init** (1).